Detailed Design Specification

ALWAYS HOMe

Team: Always Home

Project: Smart Door

Team Members:
James Lunsford
Khuong Nguyen
Juan Duarte
Jay Otterbine

Last Updated: 2/15/2014

Table of Contents

Table of Contents 2
Document Revision History 5
List of Figures 6
List of Tables 7
1. Introduction 8
2. Architecture Overview 8
2.1. Architecture Description 8
2.2. Subsystem/Module Decomposition Chart 9
2.3. Data Element Descriptions 10
2.4. System Level Producer Consumer Relationships 11
2.4.1.1 Producer Consumer Analysis 11
2.4.1.2 Producer and Consumer Findings 11
2.4.2.1 Web Producer Consumer Analysis 12
2.4.2.2 Web Producer and Consumer Findings 12
2.4.3.1 Android Producer Consumer Analysis 13
2.4.3.2 Android Producer and Consumer Findings 13
2.4.4.1 Hardware Producer Consumer Analysis 14
2.4.4.2 Hardware Producer and Consumer Findings 14
2.5. UI Layer 15
2.5.1. UI Layer Decomposition Chart 15
2.5.2. UI Layer Description 15
2.6. Processing Layer 15
2.6.1. Processing Layer Decomposition Chart 15
2.6.2. Processing Layer Description 15
2.7. Network Layer 16
2.7.1. Network Layer Decomposition Chart 16
2.7.2. Network Layer Description 16
2.8. Data Storage Layer 16
2.8.1. Data Storage Layer Decomposition Chart 16
2.8.2. Data Storage Layer Description 16
3. Detailed design Specifications 17

3.1. Overall Detailed Design principles, assumptions and trade-off parameters 17
3.1.1. Overall Detailed Design Principles: 17
3.1.2. Trade-off Parameters: 17
3.2. UI Layer 18
3.2.1. UI Layer Overview 18
3.2.1. Android GUI Subsystem 18
3.2.2. Hardware Subsystem 23
3.2.3. Web GUI Subsystem 29
3.3. Processing Layer 33
3.3.1. Processing Layer Overview 33
3.3.2. Processing Layer Data Element Description 34
3.3.3. Android Application Subsystem 34
3.3.4. Microcontroller Application Subsystem 41
3.4. Network Layer 51
3.4.1. Network Layer Overview 51
3.4.2. Network Layer Data Element Description 51
3.4.3. Video Server Subsystem 52
3.4.4. Web Application Subsystem 58
3.5. Data Storage Layer 63
3.5.1. Data Storage Layer Overview 63
3.5.2. Data Storage Layer Data Element Description 63
3.5.3. Server HDD Subsystem 63
3.5.4. Micro Controller HDD Subsystem 65
4. Quality Assurance 66
4.1. Overview 66
4.1.1. Purpose 66
4.1.2. Requirements Analysis and Review 66
4.1.3. Feasibility 66
4.1.4. Documentation 66
4.1.5. Coding Review and Analysis 67
4.2. Key Test Assumptions/Requirements/dependencies 67
4.2.1. Module/Unit Level Testing 67

4.2.2. Component Level Testing 67
4.2.3. Integration Testing 68
4.2.4. System Verification Testing 68
4.3. Brief Test Case Description for Function Testing 68
4.3.1. Expanded in Test Plan 68
5. Requirements Traceability Matrix 69
5.1. Requirements Traceability Matrix Purpose 69
5.2. System Level Requirements Traceability Matrix 70
5.2.1. System Level Requirements Traceability Matrix Analysis 70
5.2.2. System Level Requirements Traceability Matrix Findings 71
5.3. UI Layer Requirements Traceability matrix 71
5.3.1. UI Layer Requirements Traceability Matrix Analysis 72
5.3.2. UI Layer Requirements Traceability Matrix Findings 72
5.4. Processing Layer Requirements Traceability matrix 73
5.4.1. Processing Layer Requirements Traceability Matrix Analysis 73
5.4.2. Processing Layer Requirements Traceability Matrix Findings 74
5.5. Network Layer Requirements Traceability matrix 74
5.5.1. Network Layer Requirements Traceability Matrix Analysis 75
5.5.2. Network Layer Requirements Traceability Matrix Findings 75
5.6. Data Storage Layer Requirements Traceability matrix 76
5.6.1. Data Storage Layer Requirements Traceability Matrix Analysis 77
5.6.2. Data Storage Layer Requirements Traceability Matrix Findings 77
6. Acceptance Plan Assumptions Relative to Detailed Design 77
6.1. Packaging and Installation 77
6.1.1. RTMP Server installation 77
6.2. Acceptance Testing 77
6.3. Acceptance Criteria 77
7. Appendices 78
7.1. N/A 78

Document Revision History

Revision Number	Revision Date	Description	Rationale
1.0	02/17/2014	Rough Draft	First draft submitted for initial review
2.0	2/28/2014	Baseline	Revised following presentation
2.1	3/1/2014	Baseline update (corrected -jmo)	Updated TOC

List of Figures

Figure 1: Subsystem Decomposition Chart 9
Figure 2: UI Layer Decomposition Chart 15
Figure 3: Processing Layer Decomposition Chart 15
Figure 4: Network Layer Decomposition Chart 16
Figure 5: Data Storage Layer Decomposition Chart 16
Figure 6: UI Layer Overview 18
Figure 8: Hardware Subsystem Overview 24
Figure 9: Web GUI Subsystem Overview 29
Figure 10: Processing Layer Overview 33
Figure 12: Microcontroller Application Overview 42
Figure 13: Network Layer Overview 51
Figure 14: Video Server Subsystem Overview 52
Figure 9: Web Application Subsystem Overview 58
Figure 16: Data Storage Layer Overview 63
Figure 17: Server HDD Subsystem Overview 64
Figure 18: Microcontroller HDD Subsystem Overview 65

List of Tables

Table 1: System Level Data Element Description 10
Table 2: System Level Producer/Consumer Relationships 11
Table 3: UI Layer Data Element Description 18
Table 4: Processing Layer Data Element Description 34
Table 5: Network Layer Data Element Description 51
Table 6: Requirements Traceability Matrix 70
Table 7: Acceptance Criteria 77

1. Introduction

The Detailed Design Specification Document provides detailed information regarding the design aspects of the Smart Door System. Included in this document are the architecture overview, detailed design specs, layer definitions, layer subsystems and modules, and testing plan. For each section in this document, we will provide detailed information on how the Smart Door System is implemented.

2. Architecture Overview

2.1. Architecture Description

One of our design principles for the Smart Door product architecture is simplicity. We strove to develop a product architecture that is as simple as possible while still fulfilling the requirements levied upon the Smart Door system. In order to complete our architecture while achieving this goal we have developed a three layer functional architecture where the processing layer has been extended. The layers we have defined for the Smart door functional architecture are UI Layer, Process Layer, Server Layer, and Storage Layer. In addition to a functional architecture the HW component subsystem in the UI layer will include a physical architecture description. This was found to be necessary due to a large number of requirements levied on the physical design and layout of the Smart Door device. The defined layers and the subsystems within them fully support the requirements and functionality defined for the Smart Door system.

2.2. Subsystem/Module Decomposition Chart

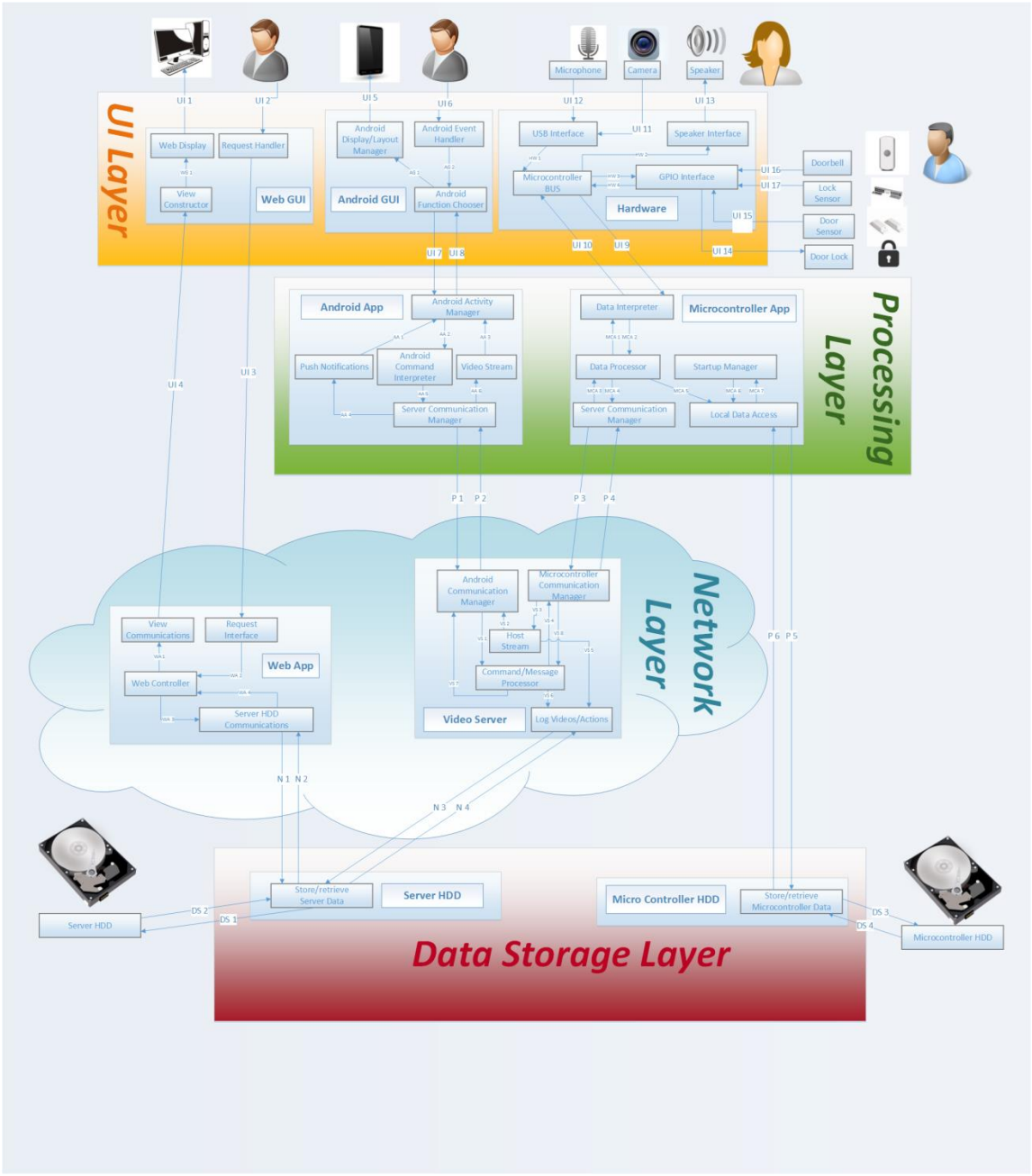


Figure 1: Subsystem Decomposition Chart

2.3.Data Element Descriptions

System Level Data Element Description	
Data Element	**Description**
UI 1	Webpages displayed on users Monitor from Web GUI
UI 2	User Input into Web GUI
UI 3	User Input passed to Web App from Web GUI
UI 4	Webpages and messages to be displayed by Web GUI from Web App
UI 5	Interactions displayed on users Android Device
UI 6	User Input into Android App
UI 7	User Input passed to Android App from Android GUI
UI 8	Interactions to be displayed by the Android GUI form the Android App
UI 9	Raw Data from the Hardware being passed to the Microcontroller App
UI 10	Audio Data and Lock/Unlock commands from Microcontroller App to the Hardware
UI 11	Raw Data from Camera to the Hardware Subsystem
UI 12	Raw Data from Microphone to the Hardware Subsystem
UI 13	Audio Stream from Hardware to User at the Door
UI 14	Lock/Unlock Signal from Hardware Subsystem to the Door Lock
UI 15	Raw Data from Door Sensor to the Hardware Subsystem
UI 16	Raw Data from Doorbell to the Hardware Subsystem
UI 17	Raw Data from Door Lock to the Hardware Subsystem
P 1	Information Requests and Commands from Android App to Video Server
P 2	Video/Audio Streams, and Sensor Information from Video Server to Android App
P 3	Video/Audio Streams, and Sensor Information from Microcontroller App to Video Server
P 4	Information Requests and Commands from Video Server to Microcontroller App
P 5	Recorded Notifications and MPG4 Files from Microcontroller App to Microcontroller HDD
P 6	Recorded Notifications and MPG4 Files from Microcontroller HDD to Microcontroller App
N 1	Information Requests from Web App to Server HDD
N 2	Recorded Notifications and MPG4 Files from Microcontroller HDD to Microcontroller App
N 3	Information Requests and MPG4 Files from Video Server to Server HDD
N 4	Recorded Notifications and MPG4 Files from Microcontroller HDD to Microcontroller App
DS 1	Recorded Notifications and MPG4 Files from Server HDD Subsystem to Physical HDD
DS 2	Recorded Notifications and MPG4 Files from Physical HDD to Server HDD Subsystem
DS 3	Recorded Notifications and MPG4 Files from Microcontroller HDD Subsystem to Physical HDD
DS 4	Recorded Notifications and MPG4 Files from Physical HDD to Microcontroller HDD Subsystem

Table 1: System Level Data Element Description

2.4. System Level Producer Consumer Relationships

Producer/Consumer Table

Consumer \ Producer	Hardware	Web GUI	Android GUI	Microcontroller App	Android App	Web App	Video Server	Microcontroller HDD	Server HDD	EXTERNAL
Hardware				UI 10						UI 11, UI 12, UI 15, UI 16, UI 17
Web GUI					UI 4					UI 2
Android GUI					UI 8					UI 6
Microcontroller App	UI 9						P 4	P 6		
Android App		UI 3	UI 7				P 2			
Web App									N 2	
Video Server				P 3	P 1				N 4	
Microcontroller HDD					P 5					DS 4
Server HDD						N 1	N 3			DS 2
External	UI 13, UI 14	UI 1	UI 5					DS 3	DS 1	

Table 2: System Level Producer/Consumer Relationships

2.4.1.1 Producer Consumer Analysis

From our ADS, we were able to analyze how the system communicates and which ones actually create what for each layer. The biggest analysis is how the table is symmetrical on each side showing that they all have a similar one to one relationship.

2.4.1.2 Producer and Consumer Findings

The main findings we found is the external hardware produces the most so we will need the communication on that layer to be able to convert the data well. The final thing is we see that the server level is where cross communication happens.

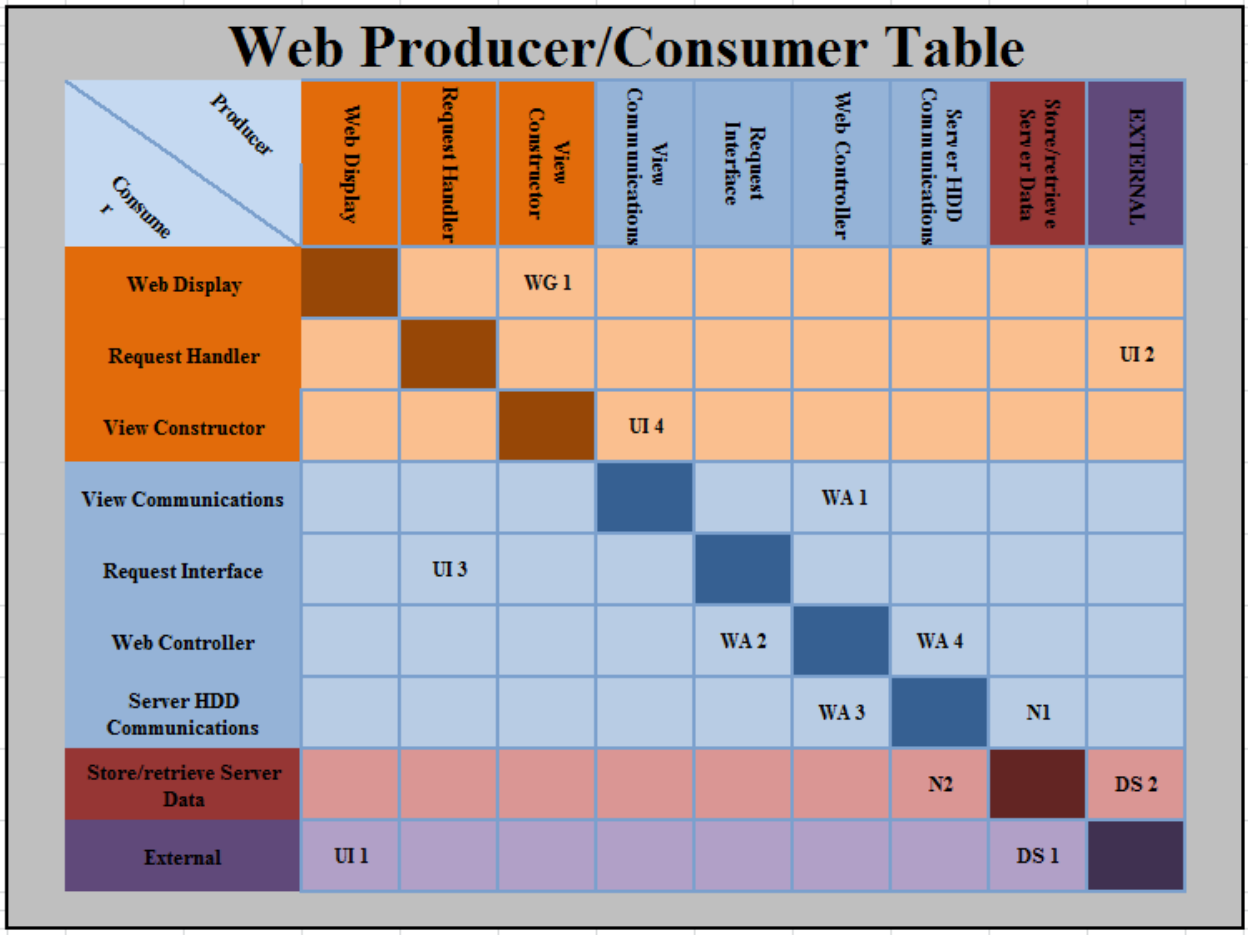


Table 3: Detailed Design View of Web Producer/Consumer Relationships

2.4.2.1 Web Producer Consumer Analysis

After comparing the mobile and hardware, the web relationships are not as complex as the other. It is the only one that doesn't have a one to one mapping. Finally, the chart doesn't show a symmetric consumer/producer side showing the items do not go in and out of the same module.

2.4.2.2 Web Producer and Consumer Findings

The main findings we found is the web is loosely coupled compared to the rest of the system. The main connection comes from the database instead of from the server layer. The other finding is there is no application layer in the web because the server is its application layer. Finally, there is only one line Web GUI line to show communication between the modules in that subsystem.

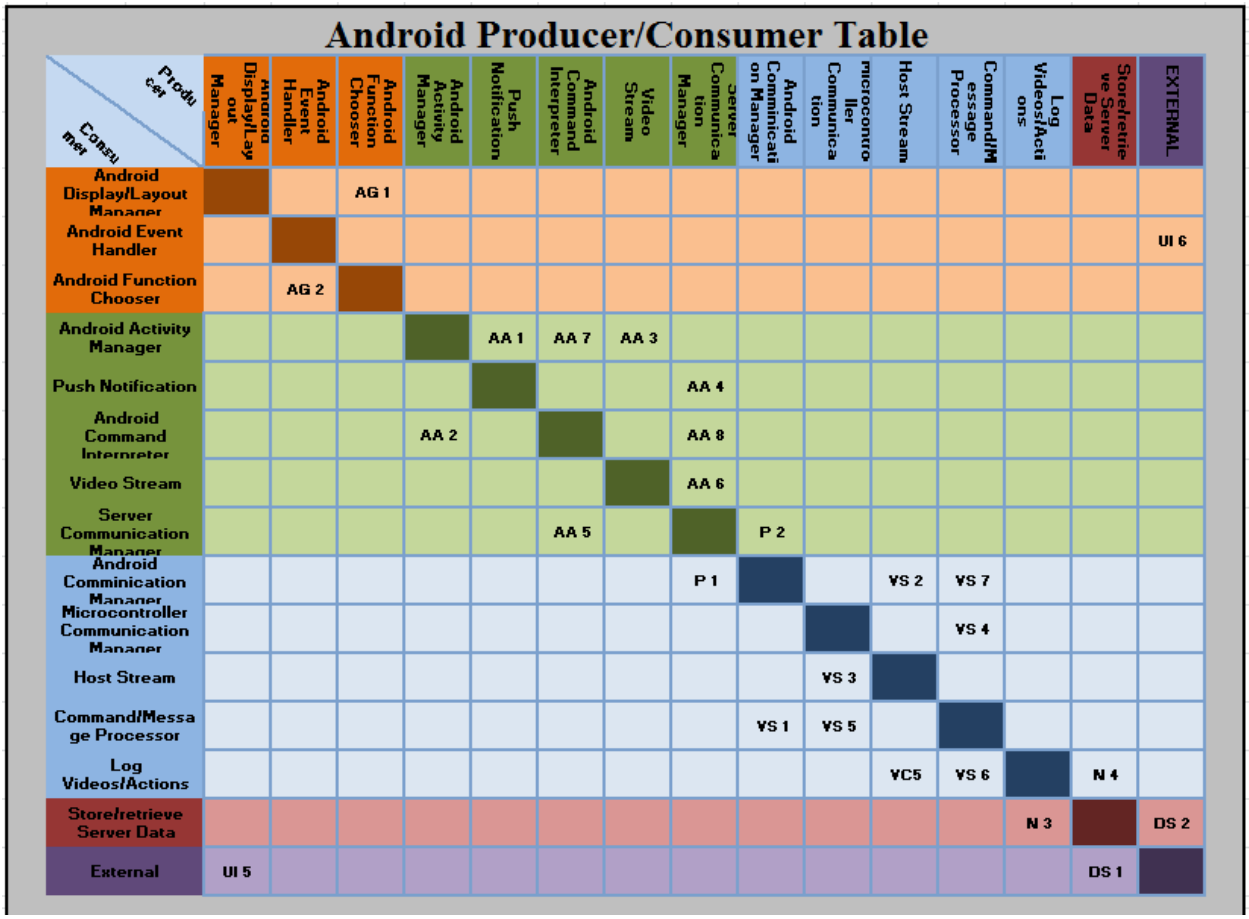


Table 4: Detailed Design View of Web Producer/Consumer Relationships

2.4.3.1 Android Producer Consumer Analysis

The Mobile analysis with this is there are more producers in the application layer because of factoring how message will be transferred for the door lock status. The next thing this shows is the server layer consumes more information because it is taking a lot more input from the mobile and hardware layer.

2.4.3.2 Android Producer and Consumer Findings

The main findings we found is the Android is heavily reliant on the server for its communication to the hardware. Without that, the android device becomes useless. The other finding is the application layer produces more because it is creating a lot things such as converting video file types and placing them into java classes

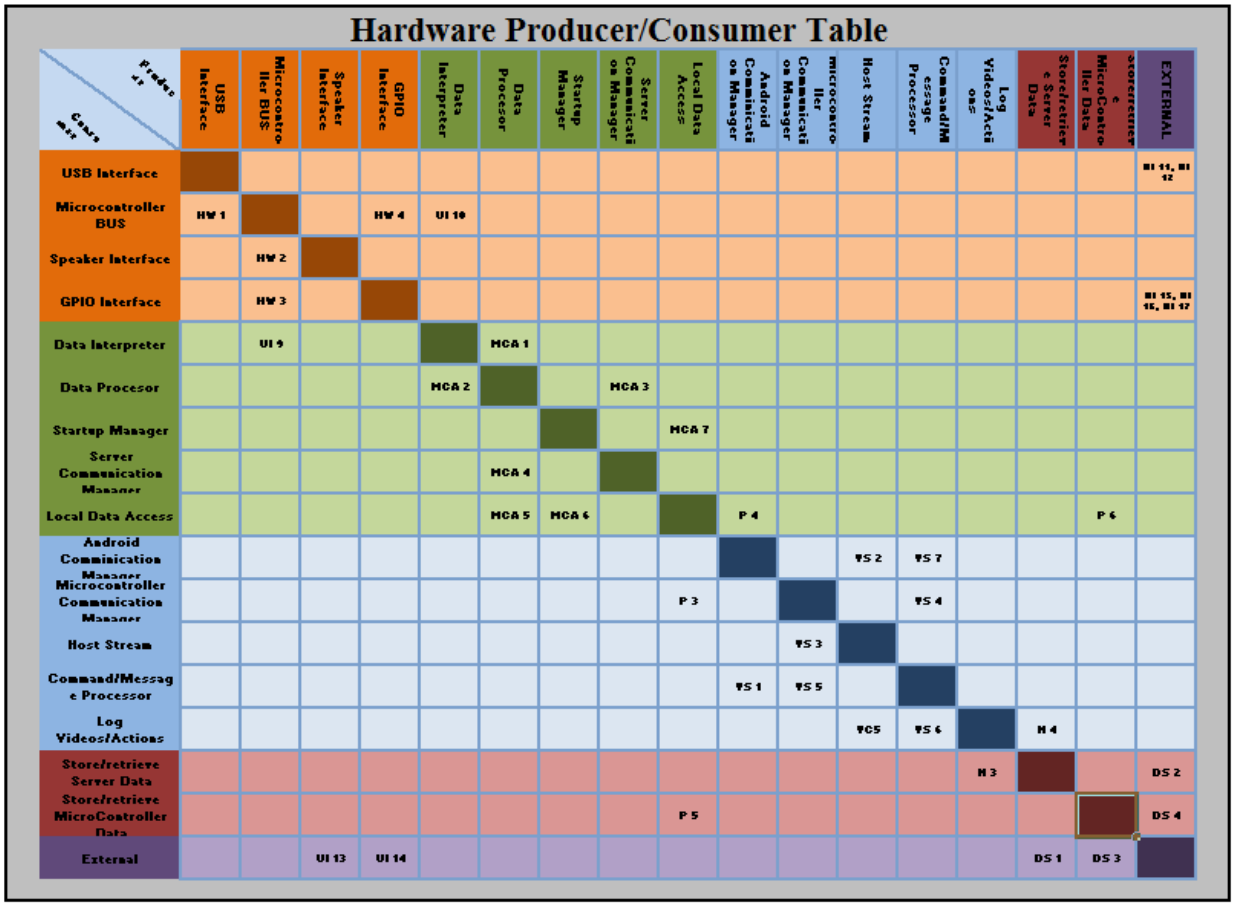


Table 5: Detailed Design View of Web Producer/Consumer Relationships

2.4.4.1 Hardware Producer Consumer Analysis

The Hardware has the most consumers and producers all due to the hardware communication. Next, the application layer deals with converting a lot of data to be sent to the server. Finally this is the only one that communicates with a local hard drive so it can store things locally if there is no internet connection.

2.4.4.2 Hardware Producer and Consumer Findings

The main findings we found is the Hardware could be our bottleneck of our system because of the raspberry pi may not be able to handle everything. The producer and consumer chart doesn't show how frequent these actions happen so it will be hard to determine if the microcontroller will be strong enough for this until we make the prototype. Finally this has the highest complexity because it involves communicating back and forth with the most modules/layers because it takes into account when there is no internet connection.

2.5. UI Layer

2.5.1. UI Layer Decomposition Chart

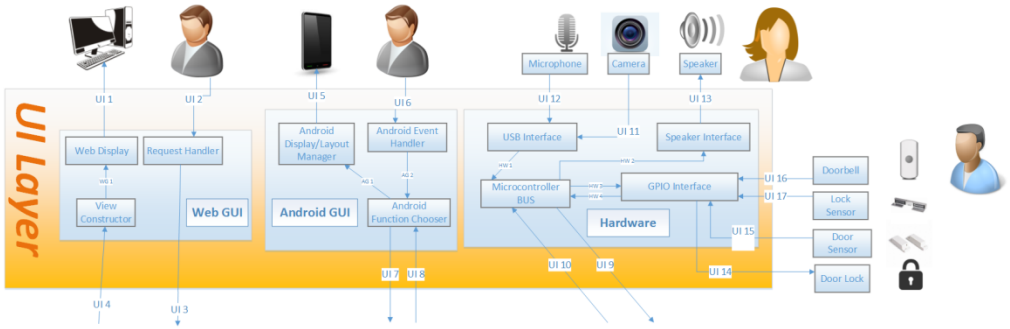


Figure 2: UI Layer Decomposition Chart

2.5.2. UI Layer Description

The UI layer receives input from the user through our software interface the Web GUI and Android GUI. It also receives input from the user through our hardware such as the doorbell, camera, and microphone. This layer will be responsible for transmitting input to other layers as well as displaying results from other layers. This will include transmitting audio and video into meaningful values to properly stream video. This layer will utilize our three different interfaces: the hardware, the web, and mobile device

2.6.Processing Layer

2.6.1. Processing Layer Decomposition Chart

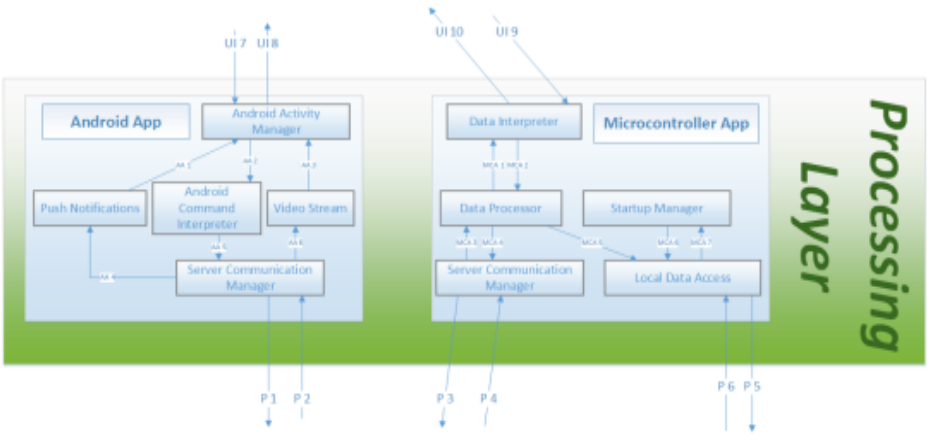


Figure 3: Processing Layer Decomposition Chart

2.6.2. Processing Layer Description

The Processing Layer will focus on converting and evaluating our data. It will perform task to take the input from the GUIs and Hardware and convert the data to information that can be transferred to other components in our system. It will have to be able to take data and prepare it for output.

2.7.Network Layer

2.7.1. Network Layer Decomposition Chart

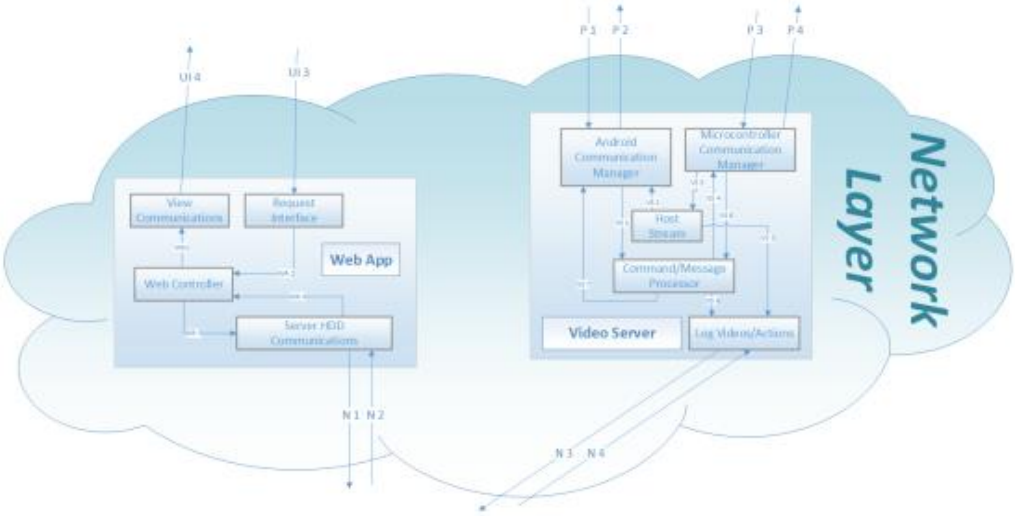


Figure 4: Network Layer Decomposition Chart

2.7.2. Network Layer Description

The Network performs tasks to convert and evaluate data in terms of information from mobile, website, and microcontroller. It also serves a way to process data from storage to meaningful information. It will perform the task of processing the different types of data from our different interface into a uniform data that can be used by other components in our system. This is the backbone of our system that will handle the bulk of the processing.

2.8.Data Storage Layer

2.8.1. Data Storage Layer Decomposition Chart

Figure 5: Data Storage Layer Decomposition Chart

2.8.2. Data Storage Layer Description

The Data Storage Layer will store information either locally or remotely on a server. This is where the information will be held until it needs to be access. This is where the storage of the videos will be for user to see past ones and live ones.

3. Detailed design Specifications

3.1. Overall Detailed Design principles, assumptions and trade-off parameters

3.1.1. Overall Detailed Design Principles:

When team Always Home was developing the Detail Design Principles, we identified seven fundamental detail design principles that contribute to the Smart Door project's successful development. These are:

- Interactive: The smart Door devices primary purpose is to facilitate interaction between two people who are separated from each other by a large physical distance. The design of the system must be transparent enough to enable personal interaction over a long distance.
- Responsiveness: Interactions will not be meaningful unless the system responds quickly enough so that it does not hinder the normal flow of conversation between users.
- Portable: The system must be designed to be portable and easy to install. This includes both the software and physical components.
- Usability: The system must be easy to set up and operate in order for users who may not be technologically sophisticated to be able to use the Smart Door product.
- Simplicity: In order to meet a large majority of our requirements the Smart Door system must be simple. Simple to set up, simple to configure, simple to operate, overall the system must be simple.
- Security: The Smart Door device must be theft resistant.

3.1.2. Trade-off Parameters:

We have identified seven trade-offs correspond to seven detailed design key principles:

- In terms of interactive, we have to keep interactive no matter what because that is the whole point of our system.
- For responsiveness, it is needed but the trade-off is how much time we will have to work on it to get a good response time. We determined the specification on a good response time in our SRS.
- Portable will make our system easy to install but this one could be optional to do. To do this, the system will be able to mount on any door, but that might not be the case because of having to power it.
- Usability and Simplicity goes hand in hand where it makes the system better experience for our users. To do this, we will have to limit what they can do with the system and minimize all user input we can.
- For our decision for the overall system, we will try to keep true to our key principles. If time for development becomes an issue, we will remove the portable principle to make it easier for us to keep it made for only one particular area.

- The last one is security of the system. These involve making a sturdy product. The trade-off is we would have to install it ourselves to make sure it is secured on the surface for security. Since we do not cover installation according to our SRS, we cannot guarantee the user will follow these principles.

3.2.UI Layer

3.2.1. UI Layer Overview

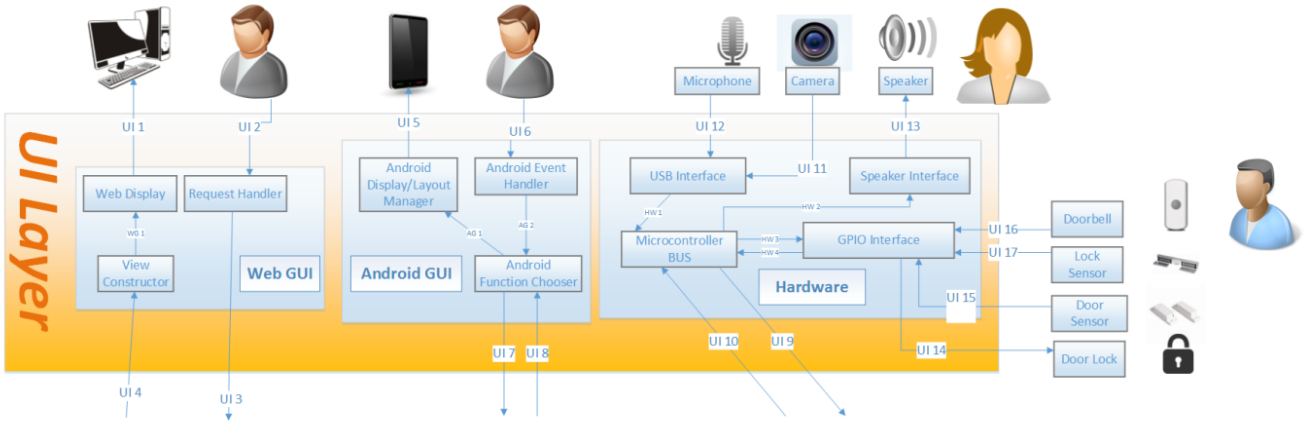


Figure 6: UI Layer Overview

3.2.1.1. UI Layer Data Element Descriptions

Subsystem Level Data Element Description	
UI Layer	
Data Element	**Description**
WG 1	HTML View Elements to display on the Users Monitor
AG 1	Java Elements to display on the Users Android Device
AG 2	Events from Action Listeners in the Android GUI
H 1	Raw Audio Signal from Microphone
H 2	Raw Audio Signal to output to Speaker
H 3	Command Signals To GPIO Interface
H 4	Status Signals from GPIO Interface

Table 6: UI Layer Data Element Description

3.2.1. Android GUI Subsystem

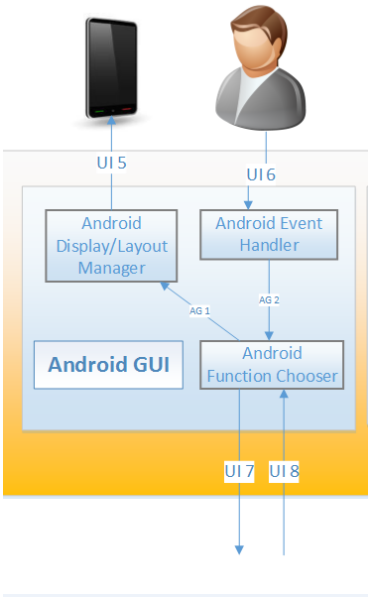
3.2.1.1. Android GUI Subsystem Principles

Communication- The Android GUI has to be interactive to allow communication between the mobile device and hardware to satisfy the key part of our project.

Usability- The Android GUI will be simple to use and have buttons for the users to click to start actions in our system. This way any user can use our application.

Responsiveness- The Android GUI will need to be responsive for the buttons in order to make users feel like the application is working.

3.2.1.2. Android GUI Subsystem Diagram



3.2.1.3. Android Display/Layout Manager Module Description

3.2.1.3.1. Android Display/Layout Manager Module Prologue

3.2.1.3.1.1. Description

The Android Display/Layout Manager will display predefined screens on the application.

3.2.1.3.1.2. Function

The primary functionality of this module is to have the following screens:

1. Login screen
2. Home screen
3. Call Screen

3.2.1.3.2. Android Display/Layout Manager Module Interfaces

Android Display/ Layout Manager I/O		
Reference #	**IN / OUT**	**Description**
UI 5	Out	Android GUI screens and buttons
AG 1	IN	Video, Audio, Door unlock/lock, push notification

3.2.1.3.3. Android Display/Layout Manager Module Physical Data Structure

Android Display/ Layout Manager Physical Data Structure		
Name	**Type**	**Description**
Video	File	This will be a live video from the microcontroller
Audio	File	This will be live audio from the microcontroller
Unlock/lock	Int	1-lock 2-unlock

3.2.1.3.4. Android Display/Layout Manager Module Processing

```
/*
 * Call Screen
 * The other 2 screens will be very similar with a layout.
 */
```

```
/*
 * Call Screen
 * The other 2 screens will be very similar with a layout.
 */

<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Smart  Door</string>
   <string name="button_end">End </string>
   <string name="button_accept">Accept  Call </string>
   <string name="button_decline">Decline  Call </string>
   <string name="title_activity_main"> Call</string>
//need Video stream here to be place on the background
</resources>

  <Button
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/button_accept" />

  <Button
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/button_decline" />

  <Button
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/button_end" />
```

3.2.1.4. Android Event Handler Module Description

3.2.1.4.1. Android Event Handler Prologue

3.2.1.4.1.1. Description

The Android Event Handler will control all the action listeners for the button when they are pressed.

3.2.1.4.1.2. Function

The primary functionality of this module is to have actionlisteners for following buttons:

- Login Screen Send
- Monitor
- Door Lock/Unlock
- Accept Call
- Decline Call
- End Call

3.2.1.4.2. Android Event Handler Interfaces

Android Event Handler I/O		
Reference #	**IN / OUT**	**Description**
UI 6	IN	User input such as Button Presses and Android System Actions
AG 2	OUT	Function Calls

3.2.1.4.3. Android Event Handler Physical Data Structure

Android Event Handler Physical Data Structure		
Name	**Type**	**Description**
OnClick	ActionListener	This will control our button when they are pressed

3.2.1.4.4. Android Event Handler Processing

```
/* each button will have a similar layout
 */
button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
           // Call function
    });
```

3.2.1.5. Android Function Chooser Description

3.2.1.5.1. Android Function Chooser Prologue

3.2.1.5.1.1. Description

The Android Function Chooser will control all the functionality of what the buttons do

3.2.1.5.1.2. Function

The primary functionality of this module is to have functions and functions that call the Android Application layers functions for following buttons:

- Login Screen Send
- Monitor
- Door Lock/Unlock
- Accept Call
- Decline Call
- End Call

3.2.1.5.2. Android Function Chooser Interfaces

Android Function Chooser I/O		
Reference #	**IN / OUT**	**Description**
UI 7	OUT	Formatted User requests
UI 8	IN	Video, data from Smart Door device
AG 2	IN	Function Calls
AG 1	OUT	Video, Audio, Door unlock/lock, push notification

3.2.1.5.3. Android Function Chooser Physical Data Structure

Android Function Chooser Physical Data Structure		
Name	**Type**	**Description**
Video	File	This will be a live video from the microcontroller
Audio	File	This will be live audio from the microcontroller
Unlock/lock	Int	1-lock2-unlock

3.2.1.5.4. Android Function Chooser Processing

```
/*
 * each button will have a similar layout for the function call
 */
```

```
Public File Monitor(){
        File video = AppMonitor(); //call the Mobile Application function
        return video; //return back to the layout view
}
```

3.2.2. Hardware Subsystem

3.2.2.1. Hardware Subsystem Description

The Hardware subsystem consists of the hardware devices in the Smart Door system and the Microcontroller BUS that accesses them. These devices include a Microphone, Camera, Speaker, Doorbell, Lock Sensor, Door Sensor and Door Lock. The functionality of this subsystem is to convert analog input into digital signals and create analog input from digital signals. This conversion is a built in function of the microcontroller so this subsystem represents how our system will access and use that functionality.

3.2.2.2. Hardware Subsystem Diagram

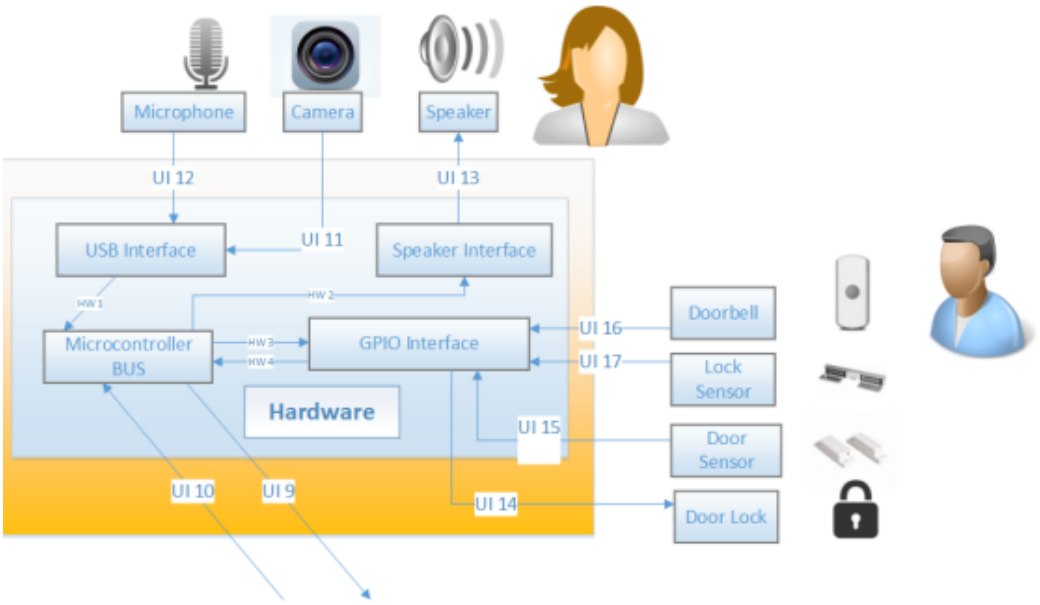


Figure 7: Hardware Subsystem Overview

3.2.2.3. Hardware Subsystem Principles

Interactive: The hardware subsystem is one of the three user visible parts of the application. The Hardware subsystem enables communication between the user and their guests through the connected devices. This subsystem represents the end of the communications chain where the analog input and output are received and transmitted. Thus the design of the system must be transparent enough to enable personal interaction over a long distance.

Responsiveness: The hardware subsystem has to function quickly in order for the system to be responsive. Any delay or failure of this system will completely disable the communication functionality of the Smart Door system.

Reliability: The overall Smart Door performance relies strongly on the reliability of the hardware subsystem. Any delay or failure of this system will completely disable the communication functionality of the Smart Door system.

3.2.2.4. USB Interface Module Description

3.2.2.4.1. USB Interface Description

The USB Interface Module is a connection interface - i.e. a USB Hub device to provide connection between the Microcontroller and Microphone, Camera.

3.2.2.4.2. USB Interface Function

The main function of the USB interface is to provide stable, 2-way communication between the Microphone, Camera and the Microcontroller.

3.2.2.4.3. USB Interface Module Interfaces

USB Interface I/O		
Reference #	**IN / OUT**	**Description**
UI 11	IN	Digital video data stream from the attached camera
UI 12	IN	Digital audio data stream from the attached microphone
HW 1	OUT	Digital video and audio data streams to be passed to Microcontroller App

3.2.2.4.4. USB Interface Module Physical Data Structure

USB Interface Physical Data Structure		
Name	**Type**	**Description**
video_In	Digital Signal	Digital video data stream from the attached camera
audio_In	Digital Signal	Digital audio data stream from the attached microphone
video_Out	Digital Signal	Digital video data streams to be passed to Microcontroller App
audio_Out	Digital Signal	Digital audio data streams to be passed to Microcontroller App

3.2.2.4.5. USB Interface Module Processing

```
/*
 * A while loop will wait for input data
 * If data is receiving, begin transmitting the data
 */
while( no digital data stream ) {
        wait and listen
        if( data is received)
                Proceed to transmit to Microcontroller.
}
```

3.2.2.5. Speaker Interface Module Description

3.2.2.5.1. Speaker Interface Description

The Speaker Interface Module is a connection interface that establishes connection between the Speaker and the Microcontroller –i.e. USB sound cards that build-in or externally plug into the Microcontroller via USB port.

3.2.2.5.2. Speaker Interface Function

The Speaker Interface, allowing a single driver to work with the various USB sound devices on the market and establishes connections between the speaker and the Microcontroller

3.2.2.5.3. Speaker Interface Module Interfaces

Speaker Interface I/O		
Reference #	**IN / OUT**	**Description**
UI 13	OUT	Digital audio data stream to be passed to the Speaker
HW 2	IN	Digital audio data streams from the Microcontroller App

3.2.2.5.4. Speaker Interface Module Physical Data Structure

Speaker Interface Physical Data Structure		
Name	**Type**	**Description**
audio_In	Digital Signal	Digital audio data stream from the Microcontroller App
audio_Out	Digital Signal	Digital audio data streams to be passed to the Speaker

3.2.2.5.5. Speaker Interface Module Processing

```
/*
  * While loop to listen for output
  */
while( no digital sound data from the Microcontroller ) {
        wait and listen
        if( digital sound data is received)
                proceed to transmit to speaker.
}
```

3.2.2.6. GPIO Interface Module Description

3.2.2.6.1. GPIO Interface Description

The General Purpose Input/output interface is a group of generic pins on a chip whose behavior (as an input or an output) can be controlled through software. The GPIO interface can be found on the microcontroller or externally plug into the Microcontroller. The pins can be programmed as input, where data from some external source such as door lock, doorbell, lock sensor, etc… are being fed into the system to be manipulated at a desired time and location.

3.2.2.6.2. GPIO Interface Function

The General Purpose Input Output interface provides an ease of access to the devices internal properties. The pins available on the GPIO interface can be programmed to be used to either accept input or provide output to external devices such as get signal from lock sensor or Open/Close sensor, etc...

3.2.2.6.3. GPIO Interface Module Interfaces

GPIO Interface I/O		
Reference #	**IN / OUT**	**Description**
UI 14	OUT	Digital command data stream from the Microcontroller App to Door Lock
UI 15	IN	Digital notification data stream from Door Sensor
UI 16	IN	Digital notification data streams from Doorbell
UI 17	IN	Digital status data stream from Lock Sensor
HW3	IN	Digital command data from the Microcontroller App
HW4	OUT	General digital data to be passed to the Microcontroller App

3.2.2.6.4. GPIO Interface Module Physical Data Structure

GPIO Interface Physical Data Structure		
Name	**Type**	**Description**
command_Out	Digital Signal	Digital command data stream from the Microcontroller App to Door Lock
Notification1_In	Digital Signal	Digital notification data streams from Door Sensor
Notification2_In	Digital Signal	Digital notification data streams from Doorbell
Status_In	Digital Signal	Digital status data stream from Lock Sensor
Command_In	Digital Signal	Digital command data stream from the Microcontroller App
Data_Out	Digital Signal	Digital data stream to be passed to the Microcontroller App

3.2.2.6.5. GPIO Interface Module Processing

```
/*
 * The GPIO Interface  will use a while loop
 * The while loop will handle messaging from connected peripherals
 */
while( no signal ) {
        wait and listen
        if(receive command message from system)
                Process and send command signal to peripherals
        if(receive data message from peripherals)
                Send status message received from peripherals to system
}
```

3.2.2.7. Microcontroller BUS Interface Module Description

3.2.2.7.1. Microcontroller BUS Interface Description

The Microcontroller BUS Interface is built in the Microcontroller. This interface establishes and utilizes different connections to other interfaces including USB Interface, Speaker Interface, and GPIO Interface.

3.2.2.7.2. Microcontroller BUS Interface Function

The Microcontroller BUS Interface serves as a bridge to connect different interfaces to the Microcontroller.

3.2.2.7.3. GPIO Interface Module Interfaces

Microcontroller BUS Interface Module		
Reference #	**IN / OUT**	**Description**
HW1	IN	Digital data stream from USB Interface
HW2	OUT	Digital data stream to Speaker Interface
HW3	OUT	Digital data stream to the GPIO Interface
HW4	IN	Digital data stream from the GPIO Interface
UI10	IN	Digital data from the Microcontroller App
UI9	OUT	Digital data to the Microcontroller App

3.2.2.7.4. GPIO Interface Module Physical Data Structure

Microcontroller BUS Interface		
Name	**Type**	**Description**
audio_In	Digital Signal	Digital audio data stream from the Microcontroller App
audio_Out	Digital Signal	Digital audio data streams to be passed to the Speaker
command_Out	Digital Signal	Digital command streams to GPIO Interface
data_In	Digital Signal	Digital data streams from the GPIO Interface
data_Out	Digital Signal	Digital data streams to be passed to the Microcontroller App
command_In	Digital Signal	Digital command streams from the Microcontroller App

3.2.2.7.5. Microcontroller BUS Interface Module Processing

```
/*
 * The Microcontroller BUS Interface  will use a while loop
 * The while loop will handle messaging from connected peripherals
 */
while( no signal ) {
        wait and listen
        if(receive command message from Microcontroller)
                Identify signal path;
                Send command streams to the right peripherals;
        if(receive data message from peripherals)
                Send data packets to Microcontroller App;
}
```

3.2.3. Web GUI Subsystem

3.2.3.1. Web GUI Subsystem Principles

Simplicity: In order to meet the Smart Door Project Overall Principles, the Web GUI must be simple to navigate, simple to configure, and simple to operate.

Complete: The Web GUI Subsystem must fully describe the Web Application Subsystem functionality

Security: The Web GUI Subsystem must be hacker resistant.

3.2.3.2. Web GUI Subsystem Diagram

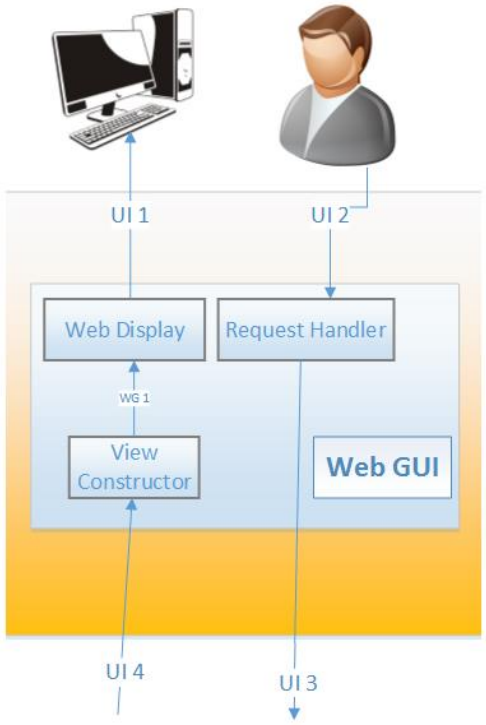


Figure 8: Web GUI Subsystem Overview

3.2.3.3. Web Display Module Description

3.2.3.3.1. Web Display Description

The Web Display is in charge of sending the compiled HTML file and the corresponding JavaScript and CSS files. It can also send video and image file to the user. It handles the transportation of the data back to the user's computer.

3.2.3.3.2. Web Display Function

The main function of the Web Display is to move the data created by the view constructer to the user that requested the data.

3.2.3.3.3. Web Display Module Interfaces

Web Display Interfaces I/O		
Reference #	**IN / OUT**	**Description**
UI1	OUT	Files streams to the users browser application.
WG1	IN	Data object containing a list of file.

3.2.3.3.4. Web Display Module Physical Data Structure

Web Display Physical Data Structure		
Name	**Type**	**Description**
HTML File	HTML	A HTML file that was created by the View Constructor
JavaScript File	JavaScript	Static JavaScript file in the Scripts folder
CSS File	CSS	Static CSS file in the Content folder
Video File	Mp4	Static video in the SQL database

3.2.3.3.5. Web Display Module Processing

```
/*
 *  A while loop that buffers data to be sent to the user
 */
while( data ) {
        buffer data
        transmit(buffer);
}
```

3.2.3.4. View Constructor Module Description

3.2.3.4.1. View Constructor Description

The View Constructor runs razor syntax code to create dynamic HTML pages. The View Constructor uses data that is passed down from the Web Application to create the content in the HTML code. The module holds HTML templates for logging in, logging out, downloading videos, pairing device, monitoring live video, listing videos, and for account configuration.

3.2.3.4.2. View Constructor Function

The main function of the View Constructor is to construct the entire HTML file, and retrieves all the necessary files for the web display to send to the user. The HTML templates are the most important part of this module because they enable the user to trigger web applications functionality.

3.2.3.4.3. View Constructor Module Interfaces

View Constructor I/O		
Reference #	**IN / OUT**	**Description**
UI4	IN	Data object with the templates the constructor need to execute
WG1	OUT	Files that the web display needs to transmit

3.2.3.4.4. View Constructor Module Physical Data Structure

View Constructor Physical Data Structure		
Name	**Type**	**Description**
Login/Logout Template	CSHTML file	The Login/Logout template allows the user to enter credentials and exit his or her account. In the logging in aspect of the module the system will take in and username and password and send it to the server. The password will be hashed using SHA-1 at the client side and the salted in the server. The module will also have a link to the registration page for new users, and a remember-me checkbox to allow the user to stay logged in by keeping his or her session alive. The log out aspect will destroy the user's session, and the user will be redirected to the public home page.
Download Video Template	CSHTML file	The Download template identifies to the server which video file the user would like to download.
Pair Device Template	CSHTML file	The pair device template allows the user to enter smart door device keys and generate mobile device key. The system can identify the owner of the device and map the calls correctly with this information.
Monitoring Live Video Template	CSHTML file	The monitor template gives the user the ability to see through the smart doors camera at any time. The module will need the user to specify which smart door device he would like to look at.
List Videos Template	CSHTML file	The video template allows the user to see when video interactions occurred between the smart door device and the mobile device. It gives a link to the video, a button to download the video and a button to delete the video. The table will have a search input so that the user can find a particular interaction quickly.
Account Configuration Template	CSHTML file	The account configuration template allows the user to change his or her password. The module also allows new users to create a new account.

3.2.3.4.5. View Constructor Module Processing

```
/*
  *  parses data from the Web App
  *  executes main template
  *     executes sub templates
  *  Send generated file to the web display
  */

ViewConstructor ( data ) {
          dataList = getDataObjects(data)
          htmlfile = ""
          for each file execution in datalist
            run interpreter or executefile
          add results to htmlfile
        WebDisplay(htmlfile);
}
```

3.2.3.5. Request Handler Module Description

3.2.3.5.1. Request Handler Description

The Request Handler forwards the HTTP request to the Web Application

3.2.3.5.2. Request Handler Function

The main function of the Request Handler is to pass the http request if it follows the correct syntax.

3.2.3.5.3. Request Handler Module Interfaces

Request Handler I/O		
Reference #	**IN / OUT**	**Description**
UI2	IN	HTTP request from the user browser
UI3	OUT	Valid HTTP Request

3.2.3.5.4. Request Handler Module Physical Data Structure

Request Handler Data		
Name	**Type**	**Description**
Request	HTTP	A String containing a request for the Web Application

3.2.3.5.5. Request Handler Module Processing

```
/*
 * regular expression
 * send request
 */
RequestHandler( http ) {
        regularExpression = http regex
        if( match(regularExpression, http) ){
         WebApp(http);
         }
}
```

3.3.Processing Layer

3.3.1. Processing Layer Overview

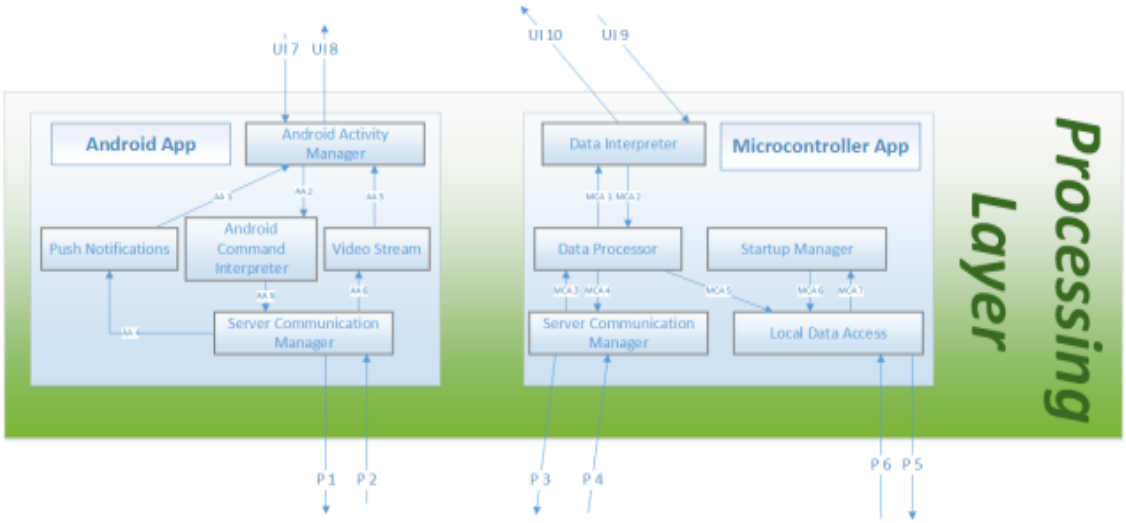


Figure 9: Processing Layer Overview

3.3.2. Processing Layer Data Element Description

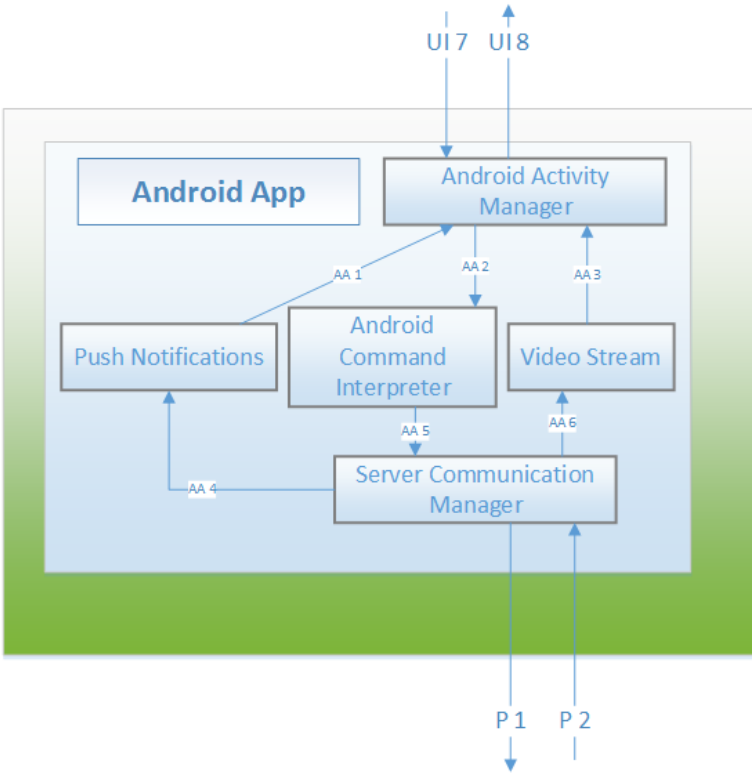
Subsystem Level Data Element Description	
Processing Layer	
Data Element	**Description**
AA 1	Push Notifications to be displayed on the Android GUI
AA 2	Events from Android GUI
AA 3	h.264 Stream from Video Server
AA 4	Messages from Video Server to be processed into Push Notifications
AA 5	Formatted User Commands from Events
AA 6	h.264 Video from Video Server
MCA 1	Formatted User Commands from Video Server
MCA 2	Formatted Input Data from Hardware Subsystem
MCA 3	User Commands from Video Server
MCA 4	h.264 Stream, and Event Messages to Video Server
MCA 5	h.264 Stream, and Event Messages to be stored on Microcontroller HDD
MCA 6	Startup Messages
MCA 7	Startup Protocols
MCA 8	Stored Videos and Notifications

Table 7: Processing Layer Data Element Description

3.3.3. Android Application Subsystem

3.3.3.1. Android Application Subsystem Principles

3.3.3.2. Android Application Subsystem Diagram



3.3.3.3. Android Activity Manager Module Description

3.3.3.3.1. Android Activity Manager Module Prologue

3.3.3.3.1.1. Description

The Android Activity Manager Module will package the output and unpackage the input from the Android GUI

3.3.3.3.1.2. Function

The main function of this will be getting the classes of the door lock, video, and audio.

3.3.3.3.2. Android Activity Manager Module Interfaces

Android Activity Manager I/O		
Reference #	**IN / OUT**	**Description**
UI 7	OUT	h.264 Stream, Push Notifications, System Status
UI 8	IN	User Input
AA 1	IN	Push Notifications
AA 2	OUT	User Input
AA 3	IN	h.264 Stream

3.3.3.3.3. Android Activity Manager Module Physical Data Structure

Android Activity Manager Physical Data Structure		
Name	**Type**	**Description**
Video	File	This will be a live video from the microcontroller
Audio	File	This will be live audio from the microcontroller
Unlock/lock	Int	1-lock 2-unlock

3.3.3.3.4. Android Activity Manager Module Processing

```
/*
 * This will be getting things from the classes and passing it
 */
AndroidProcessingLayer(){
                    //this will call the getters of the classes.
}
```

3.3.3.4. Server Communication Manager Module Description

3.3.3.4.1. Server Communication Manager Module Prologue

3.3.3.4.1.1. Description

The Server Manager Module will communicate with the server to be able to communicate with the microcontroller.

3.3.3.4.1.2. Function

The main function of this is to be able to pull the video and audio feeds as well as the status of the door lock.

3.3.3.4.2. Server Communication Manager Module Interfaces

Server Communication Manager I/O		
Reference #	**IN / OUT**	**Description**
P1	OUT	User Input
P2	IN	h.264 Stream, Push Notifications, System Status
AA 4	OUT	Push Notifications
AA 5	IN	User Input
AA 6	OUT	h.264 Stream

3.3.3.4.3. Server Communication Manager Module Physical Data Structure

Server Communication Manager Physical Data Structure		
Name	**Type**	**Description**
Video	h.264	This will be a live video from the microcontroller
Audio	h.264	This will be live audio from the microcontroller
Unlock/lock	Int	1-lock 2-unlock

3.3.3.4.4. Server Communication Manager Module Processing

```
/*
 * This will be getting video and audio from the server as well as door atatus
 */
Int ServerUnlockLock(){
        //this will return the status of the door
}

ServerVideoAudio(){
        //this will call the set in the video and audio class
}
```

3.3.3.5. Android Command Interpreter Module Description

3.3.3.5.1. Android Command Interpreter Door Module Prologue

3.3.3.5.1.1. Description

The Android Command Interpreter will handle the messages that are passed back and forth on the mobile device such Unlock/lock and it will also pass information down to the server.

3.3.3.5.1.2. Function

The primary functionality of this module is know if the door is locked/unlock and to be able pass any other messages such as requesting things from the server.

3.3.3.5.2. Android Command Interpreter Door Module Interfaces

Android Command Interpreter I/O		
Reference #	**IN / OUT**	**Description**
AA 2	IN	User Input
AA 5	OUT	User Input

3.3.3.5.3. Android Command Interpreter Module Physical Data Structure

Android Command Interpreter Physical Data Structure		
Name	**Type**	**Description**
Unlock/lock	Int	1-lock 2-unlock

3.3.3.5.4. Android Command Interpreter Module Processing

```
/*
 * This will be getting messages from the server
 */
AndroidRequest(){
        //this will ask the server to start the microcontroller for processes
}

Int UnlockLockMessage(){
        //this will return a type int of the door being unlocked or locked.
}
```

3.3.3.6. Push Notifications Module Description

3.3.3.6.1. Push Notification Module Prologue

3.3.3.6.1.1. Description

The Push Notification Module will handle the events of what to be pushed and presented to the UI

3.3.3.6.1.2. Function

The primary functionality of this module is to choose the events of either incoming call or missed call. This will handle it be able to prepare for presentation.

3.3.3.6.2. Push Notification Module Interfaces

Push Notifications I/O		
Reference #	**IN / OUT**	**Description**
AA 1	OUT	Push Notifications
AA 4	IN	Push Notifications

3.3.3.6.3. Push Notification Module Data Structure

Push Notifications Physical Data Structure		
Name	**Type**	**Description**
Push	pushNotification	These are native to the android phone and will be when you have a call or missed call

3.3.3.6.4. Push Notification Module Processing

```
receivedEvent: function(id) {
      var pushNotification = window.plugins.pushNotification;
      if (device.platform == 'android' || device.platform == 'Android') {
         pushNotification.register(successHandler,
errorHandler,{"senderID":"834841663931","ecb":"onNotificationGCM"});
      }
      else {
         pushNotification.register(this.tokenHandler,this.errorHandler,
{"badge":"true","sound":"true","alert":"true","ecb":"app.onNotificationAPN"});
      }
...
   }

} //provided by phonegap exampleActivity
```

3.3.3.6.5. Video Stream Module Description

3.3.3.6.5.1. Description

Video Stream Module will pull a video and audio stream that is posted to the server.

3.3.3.6.5.2. Function

The primary functionality of this module is to allow other subsystems in the mobile app to have access to a video and audio stream.

3.3.3.6.6. Video Stream Module Interfaces

Video Stream Manager I/O		
#	IN / OUT	Description
AA 3	OUT	h.264 Stream
AA 6	IN	h.264 Stream

3.3.3.6.7. Video Stream Module Physical Data Structure

Video Stream Physical Data Structure		
Name	Type	Description
Video	File	This will be a live video from the microcontroller
Audio	File	This will be live audio from the microcontroller

3.3.3.6.8. Video Stream Module Processing

```
/*
 * This is sample code that is provided by android device for media
 */

/**
 * Creates a media file in the {@code Environment.DIRECTORY_PICTURES}
 * directory.
 * The directory is persistent and available to other applications like
 * gallery.
 *
 * @param type Media type. Can be video or image.
 * @return A file object pointing to the newly created file.
 */
    public  static File getOutputMediaFile(int type){
        // To be 12safe, you should check that the SDCard is mounted
        // using Environment.getExternalStorageState() before doing this.
        if
(!Environment.getExternalStorageState().equalsIgnoreCase(Environment.MEDIA_MO
UNTED)) {
            return  null;
        }

        File mediaStorageDir = new
File(Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES), "CameraSample");
        // This location works best if you want the created images to be
shared
        // between applications and persist after your app has been
uninstalled.
```

```
        // Create the storage directory if it does not exist
        if (! mediaStorageDir.exists()){
            if (! mediaStorageDir.mkdirs()) {
                Log.d("CameraSample", "failed to create directory");
                return null;
            }
        }

        // Create a media file name
        String timeStamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        File mediaFile;
        if (type == MEDIA_TYPE_IMAGE){
            mediaFile = new File(mediaStorageDir.getPath() + File.separator
+
                    "IMG_"+ timeStamp + ".jpg");
        } else if(type == MEDIA_TYPE_VIDEO) {
            mediaFile = new File(mediaStorageDir.getPath() + File.separator
+
                    "VID_"+ timeStamp + ".mp4");
        } else {
            return null;
        }

        return mediaFile;
}
```

3.3.4. Microcontroller Application Subsystem

3.3.4.1. Microcontroller Application Subsystem Description

The Microcontroller Application subsystem is responsible for all the message passing, stream encoding and startup procedures of the microcontroller. Always Home has generalized this component as an application but some of the startup procedures will actually be initialized with registry keys and the streaming will use the Linux library avconv from Libav. The command messages, stream initialization and video storage functionalities will be controlled by the custom C++ application Always Home has designed.

3.3.4.2. Microcontroller Application Subsystem Diagram

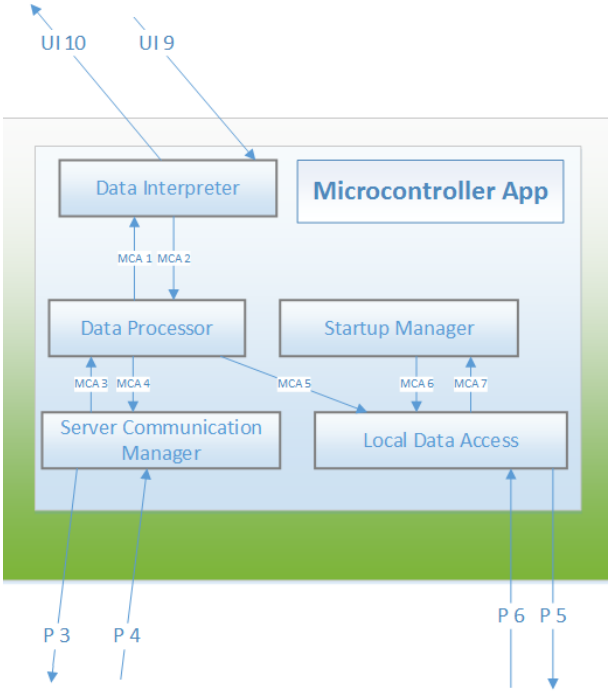


Figure 10: Microcontroller Application Overview

3.3.4.3. Microcontroller Application Subsystem Principles

Interactive: The Microcontroller Application Subsystem supports the interactive principle by facilitating the communication between users of the Smart Door. The Microcontroller Application Subsystem will convert raw video data into a usable h.264 stream and pass it to the Video Server Subsystem.

Responsiveness: The Microcontroller Application Subsystem supports the Responsiveness principle by translating inputs from the Hardware subsystem and interacting with the Video Server subsystem. The Microcontroller Application Subsystem will interpret inputs from the door mounted hardware and send notifications to the Video Server so that the Video Server can notify devices. Additionally, the Microcontroller Application Subsystem will turn on the camera and microphone for monitoring upon receiving a signal from the Android Application via the Video Server.

Usability: The Microcontroller Application Subsystem supports usability by enabling communication between users of the smart door system. Communication is the core functionality of the smart door system and it would not be possible without the Microcontroller Application Subsystem.

Communication: The Microcontroller Application Subsystem supports the communication principle by providing the physical link between users of the smart door system. Without the Microcontroller Application Subsystem there would be no

way for the person at the door to communicate remotely with the owner of the house.

Requirement Satisfaction: The Microcontroller Application Subsystem supports requirements satisfaction by providing a framework for the functionality described in the SRS document.

Usability: The Microcontroller Application Subsystem supports usability by providing the necessary communication between layers to support designed functionality of the system.

Reliability: The Microcontroller Application Subsystem supports Reliability by enabling the automatic launch of the system program upon booting the microcontroller.

3.3.4.4. Data Interpreter Module Description

3.3.4.4.1. Data Interpreter Description

Data Interpreter module is the software side of the hardware interface to devices controlled by the Microcontroller BUS.

3.3.4.4.2. Data Interpreter Function

The Data Interpreter module accepts three types of digital signals from the Hardware subsystem. The sensor signals are converted into Boolean values to be used for decisions by the software logic in the Data Processor module. The audio and video signals are simply passed to the Data Processor module to be converted and routed.

3.3.4.4.3. Data Interpreter Interfaces

Data Interpreter		
Reference #	**IN / OUT**	**Description**
UI 9	IN	Sensor, Audio and Video Signals from Microcontroller BUS
UI 10	OUT	Hardware Commands and Audio Stream to Microcontroller BUS
MCA 1	IN	Formatted Hardware Command Messages and Audio Stream from Data Processor
MCA 2	OUT	Formatted Sensor Signals, RAW Audio and Video Stream from Microcontroller BUS

3.3.4.4.4. Data Interpreter Physical Data Structure

Data Interpreter		
Name	**Type**	**Description**
raw_Video	Digital Signal	Digital video data stream from the Hardware subsystem
raw_Audio	Digital Signal	Digital audio data stream from the Hardware subsystem
lock_Signal	Digital Signal	Signal to activate the lock
door_State	Boolean	Variable to represent the open/close of the Door based on signal received from Hardware Subsystem
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock based on signal received from Hardware Subsystem
audio_Stream	.mp3 stream	Digital audio streams to be passed to the Hardware subsystem

3.3.4.4.5. Data Interpreter Module Processing

```
/*
  * Event Listeners will wait for any data
  * If an event is received  begin transmitting the data
  * Example code is adapted from msdn.microsoft.com
  */
class dataInterpreter {
public:
   void audioListener(digital audio signal) {

      action(pass the signal to the data processor module);
   }

   void videoListener(digital video signal) {
      action(pass the signal to the data processor module);
   }

   void doorbellListener(digital sensor signal) {
      action(pass the signal to the data processor module);
   }

   void hookEvent(CSource* pSource) {
      __hook(&CSource::MyEvent, pSource, &CReceiver:: audioListener);
      __hook(&CSource::MyEvent, pSource, &CReceiver:: videoListener);
__hook(&CSource::MyEvent, pSource, &CReceiver:: doorbellListener);
   }

   void unhookEvent(CSource* pSource) {
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
audioListener);
```

```
        __unhook(&CSource::MyEvent, pSource, &CReceiver::
audioListener);
        __unhook(&CSource::MyEvent, pSource, &CReceiver::
doorbellListener);
    }
};
```

3.3.4.5. Data Processor Module Description

3.3.4.5.1. Data Processor Description

The Data Processor module it responsible for routing all information flowing between the video server and the microcontroller. The Data Processor module will be part of the C++ application running on the microcontroller.

3.3.4.5.2. Data Processor Function

The Data Processor module will route control messages to and from the microcontroller. Depending on network availability the Data Processor module will route video and notifications to either the video server or Microcontroller HDD.

3.3.4.5.3. Data Processor Interfaces

Data Processor I/O		
Reference #	**IN / OUT**	**Description**
MCA 1	OUT	Formatted Hardware Command Messages and Audio Stream to Data Interpreter
MCA 2	IN	Sensor Signals variables, RAW Audio and Video Stream from Data Interpreter
MCA 3	IN	Control messages and .mp3 Audio stream from Server Communication Manager
MCA 4	OUT	Sensor state variables and Audio/Video Streams to Server Communication Manager
MCA 5	OUT	Sensor state variables and Audio/Video Streams to Local Data Access

3.3.4.5.4. Data Processor Physical Data Structure

Data Processor Physical Data Structure		
Name	**Type**	**Description**
lock_Signal	Integer	Integer used to represent the signal to activate the lock
mpg4_Video	.mpg4 File	Digital video data stream from the Microcontroller HDD
h624_Stream	h.264 Stream	Digital audio data stream from the Hardware subsystem
mp3_Audio	.mp3 Stream	Audio stream from users android device
door_State	Boolean	Variable to represent the open/close of the Door
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock
audio_Stream	.mp3 stream	Digital audio streams to be passed to the Hardware subsystem
server_Status	Boolean	Flag for WIFI connection status

3.3.4.5.5. Data Processor Processing

```
/*
  * Event Listeners will wait for any data
  * If an event is received  begin transmitting the data
  * Example code is adapted from msdn.microsoft.com
  */
class dataProcessor {
public:

// handle A/V streams
   void audioVisualListener(digital A/V signal) {
        if(network is available)
           {
            Encode the A/V signal as a h.264 stream
            action(pass the h.264 to the server communication
manager);
             }
        else
           {
            Encode the A/V signal as a MPG4 stream
            action(pass the MPG4 stream to the local data access
module);
            }
   }

// handle user interaction at door
void doorbellListener(digital sensor signal) {
        if(network is available)
            action(pass the signal to the server communication
manager);
        else
            action(pass the signal to the local data access
module);
   }


   void hookEvent(CSource* pSource) {
      __hook(&CSource::MyEvent, pSource, &CReceiver::
audioVisualListener);
__hook(&CSource::MyEvent, pSource, &CReceiver:: doorbellListener);
   }

   void unhookEvent(CSource* pSource) {
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
audioVisualListener);
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
doorbellListener);
```

```
    }
};
```

3.3.4.6. Server Communication Manager Module Description

3.3.4.6.1. Server Communication Manager Description

Server Communication Manager Module Manager is the portion of the C++ application responsible for communications with the Video Server Subsystem.

3.3.4.6.2. Server Communication Manager Function

Server Communication Manager Module Manager will package and transmit data to the server.

3.3.4.6.3. Server Communication Manager Module Interfaces

Server Communication Manager		
Reference #	**IN / OUT**	**Description**
MCA 3	OUT	Control messages and .mp3 Audio stream from Video Server Subsystem
MCA 4	IN	Sensor state variables and Audio/Video Streams to upload to Video Server
MCA 8	IN	Recorded Control messages and MPG4 Video streams to upload on Video Server
P3	OUT	Sensor state variables and Audio/Video Streams to upload to Video Server
P4	IN	Control messages and .mp3 Audio stream from Video Server Subsystem

3.3.4.6.4. Server Communication Manager Module Physical Data Structure

Server Communication		
Name	**Type**	**Description**
lock_Signal	Integer	Integer used to represent the signal to activate the lock
mpg4_Video	.mpg4 File	Digital video data stream from the Microcontroller HDD
h624_Stream	h.264 Stream	Digital audio data stream from the Hardware subsystem
mp3_Audio	.mp3 Stream	Audio stream from users android device
door_State	Boolean	Variable to represent the open/close of the Door
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock
audio_Stream	.mp3 stream	Digital audio streams to be passed to the Hardware subsystem

3.3.4.6.5. Server Communication Manager Module Processing

```
/*
  * Event Listeners will wait for any data
  * If an event is received  begin transmitting the data
  * Example code is adapted from msdn.microsoft.com
  */
class serverCommunication {
public:

// handle A/V streams
   void audioVisualListener(digital A/V event) {
            action(pass the h.264 stream to the Video Server)
   }

// handle user interaction at door
void doorbellListener(doorbell event) {
            action(pass the signal to the local data access
module);
   }

   void hookEvent(CSource* pSource) {
      __hook(&CSource::MyEvent, pSource, &CReceiver::
audioVisualListener);
__hook(&CSource::MyEvent, pSource, &CReceiver:: doorbellListener);
   }

   void unhookEvent(CSource* pSource) {
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
audioVisualListener);
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
doorbellListener);
   }
};
```

3.3.4.7. Local Data Access Module Description

3.3.4.7.1. Local Data Access Description

Local Data Access module is a portion of the C++ application.

3.3.4.7.2. Local Data Access Function

The Local Data Access module is responsible for storing and retrieving local data. Local data will include video streams and interaction records that are recorded by the system when there is no network present. Other data is boot record and system information required by the Startup Manager module.

3.3.4.7.3. Local Data Access Module Interfaces

Local Data Access I/O		
Reference #	**IN / OUT**	**Description**
MCA 5	IN	MPG4 Files and notifications from the Data Processor
MCA 6	IN	Boot logging Information
MCA 7	OUT	Boot procedures, registry entries from the Microcontroller HDD
MCA 8	OUT	Recorded MPG4 Files and notifications to store on the Video Server HDD
P 5	OUT	MPG4 Files and notifications to store on the Microcontroller HDD
P 6	IN	Control messages and .mp3 Audio stream from Video Server Subsystem

3.3.4.7.4. Local Data Access Module Physical Data Structure

Local Data Access		
Name	**Type**	**Description**
lock_Signal	Integer	Integer used to represent the signal to activate the lock
mpg4_Video	.mpg4 File	Digital video data stream from the Microncontroller HDD
h624_Stream	h.264 Stream	Digital audio data stream from the Hardware subsytem
mp3_Audio	.mp3 Stream	Audio stream from users android device
door_State	Boolean	Variable to represent the open/close of the Door
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock
audio_Stream	.mp3 stream	Digital audio streams to be passed to the Hardware subsystem
reg_Entries	String[n]	Registry entries
boot_Info	String[n]	System boot information

3.3.4.7.5. Local Data Access Module Processing

```
/*
  * Event Listeners will wait for any data
  * If an event is received  begin transmitting the data
  * Example code is adapted from msdn.microsoft.com
  */
class localDataAccess {
public:

// handle A/V streams
   void mpg4Listener(digital A/V event) {
            action(pass the MPG4 file to the Microcontroller HDD)
   }

// handle user interaction at door
void notificationListener(notification logs) {
            action(pass the signal to the Microcontroller HDD);
   }
```

```
   void hookEvent(CSource* pSource) {
      __hook(&CSource::MyEvent, pSource, &CReceiver::
mpg4Listener);
__hook(&CSource::MyEvent, pSource, &CReceiver::
notificationListenerr);
   }

   void unhookEvent(CSource* pSource) {
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
mpg4Listener);
      __unhook(&CSource::MyEvent, pSource, &CReceiver::
notificationListener);
   }
};
```

3.3.4.8. Startup Manager Module Description

3.3.4.8.1. Startup Manager Description

The Startup Manager module is responsible for the system startup procedure of the microcontroller. The Startup Manager module will consist of one or a series of registry entries to control how the microcontroller boots and what programs are run at startup.

3.3.4.8.2. Startup Manager Function

The Boot and Crash Recovery module will enable the Smart Door app to be run at start up and communication with the server to be established.

3.3.4.8.3. Startup Manager Interfaces

Startup Manager		
Reference #	**IN / OUT**	**Description**
MCA 6	OUT	Boot logging Information
MCA 7	IN	Boot procedures, registry entries from the Microcontroller HDD

3.3.4.8.4. Startup Manager Module Physical Data Structure

Startup Manager		
Name	**Type**	**Description**
reg_Entries	String[n]	Registry entries
boot_Info	String[n]	System boot information

3.3.4.8.5. Startup Manager Module Processing

The startup manager will not have any code to run. The registry entries that it represents will cause the C++ application to begin running.

3.4.Network Layer

3.4.1. Network Layer Overview

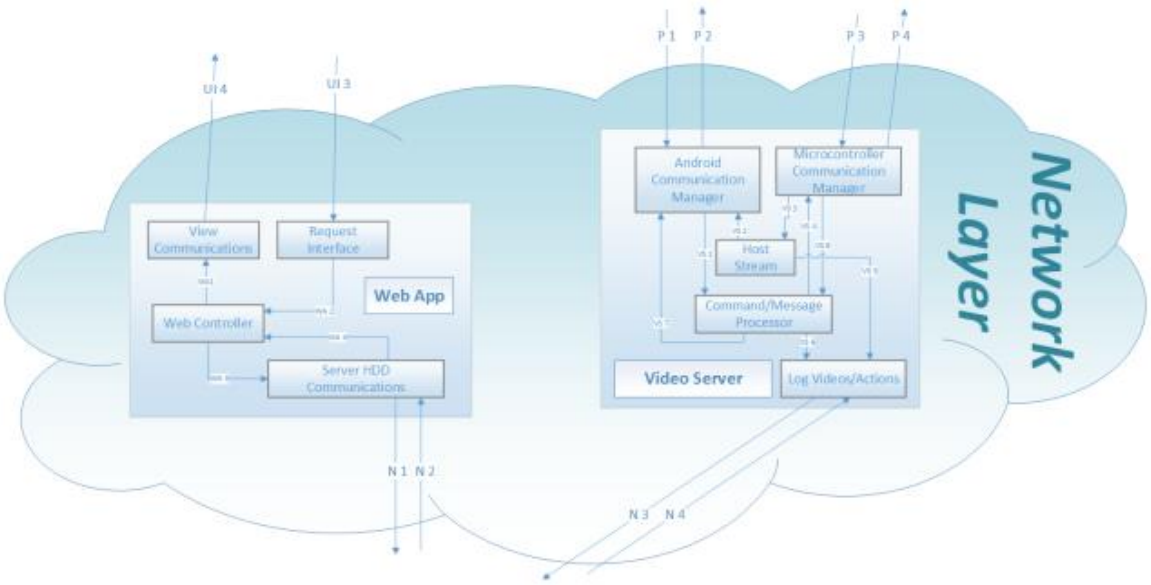


Figure 11: Network Layer Overview

3.4.2. Network Layer Data Element Description

Subsystem Level Data Element Description	
Network Layer	
Data Element	**Description**
WA 1	View Objects to forward to the Web GUI
WA 2	User Web Requests
WA 3	Requests for User Data
WA 4	User Data from Server HDD
VS 1	User Commands from Android App
VS 2	H264 Stream to be passed to Android App
VS 3	H264 Stream to be passed to Android App
VS 4	User Commands from Android App
VS 5	MPG4 Video to be Stored on Server HDD
VS 6	Notifications to be logged on Server HDD
VS 7	Notifications to be passed to Android App
VS 8	Notifications to be logged on Server HDD and passed to Android App

Table 8: Network Layer Data Element Description

3.4.3. Video Server Subsystem

3.4.3.1. Video Server Module Description

The Video Server Subsystem is facilitates interaction between the Smart Door device and the users android phone. To accomplish this goal the Video Server hosts h.264 streaming video, converts the h.264 video into MPG4 for storage, passes hardware information messages and processes commands from the Android.

3.4.3.2. Video Server Diagram

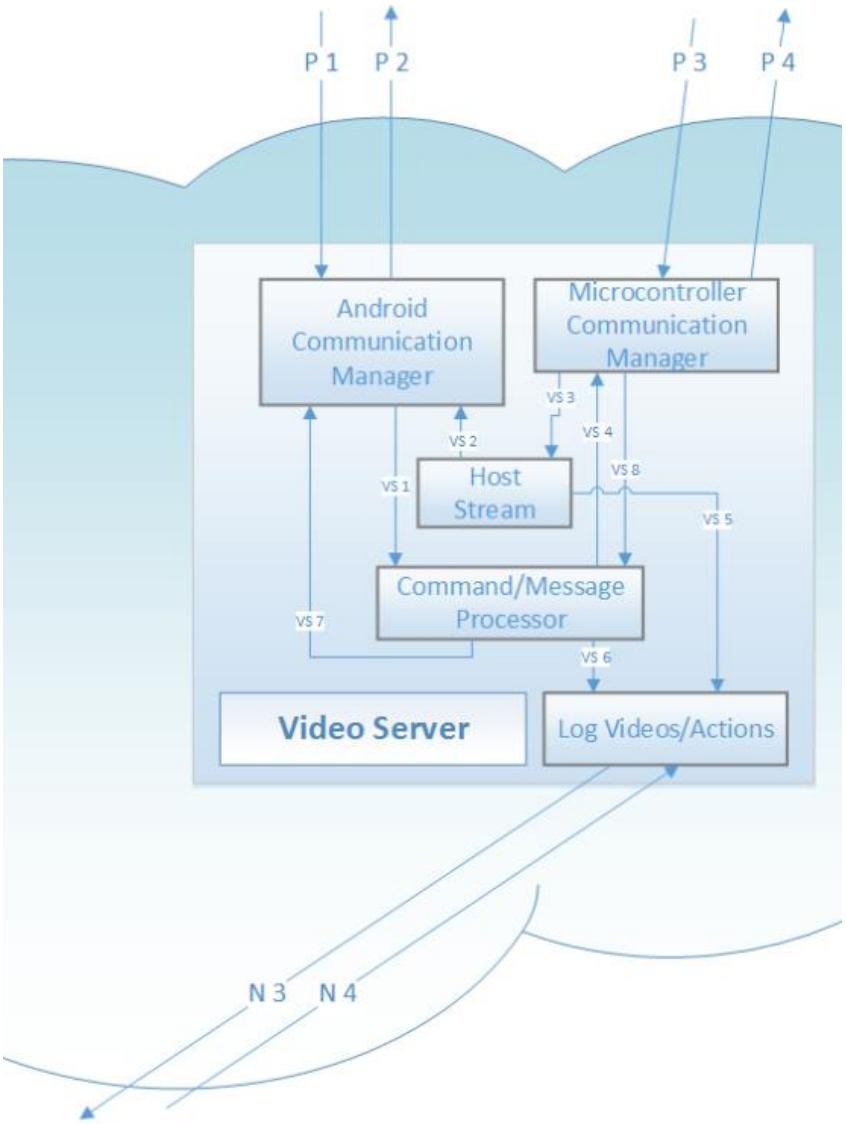


Figure 12: Video Server Subsystem Overview

3.4.3.3. Video Server Subsystem Principles

Interactive: The Video Server subsystem is a key component that facilitates interaction between the user and guests at their door by establishing a link between the Smart Door Device and the user's Android device. Interaction between the two end users would be impossible without the Video Server Subsystem to link the two users over the internet.

Responsiveness: The overall System cannot be responsive unless the Video Server operates quickly. Since this subsystem links the two end users any slow response here will be detrimental to the user experience.

Reliability: The overall Smart Door System can only be as reliable as the Video Server subsystem. Any interruption of service by the Video Server will bring down the entire system.

Communication: The Video Server subsystem is one of the critical components supporting communication functionality within the Smart Door System. Without the Video Server subsystem communication is impossible.

3.4.3.4. Host Stream Module Description

3.4.3.4.1. Host Stream Description

Host Stream Module will enable the microcontroller to stream to the Video Server. This module is a small Real Time Messaging Protocol (RTMP) server set up to run on the end users home computer.

3.4.3.4.2. Host Stream Function

The Primary functionality of this module is to make a H.264 video stream available to view by the user's android device and to transcode the stream into an mpg3 file for viewing and downloading.

3.4.3.4.3. Host Stream Module Interfaces

Host Stream		
Reference #	**IN / OUT**	**Description**
VS 2	OUT	h.264 Stream or MPG4 Stream to Android Communication Manager module
VS 3	IN	h.264 Stream from Microcontroller Communication Manager module
VS 5	OUT	MPG4 Streams to Log Videos/Actions module
VS 9	IN	MPG4 Streams from Log Videos/Actions module

3.4.3.4.4. Host Stream Module Physical Data Structure

Host Stream		
Name	**Type**	**Description**
h624_Stream	h.264 Stream	Digital Video data stream
mpg4_Video	.mpg4 File	Digital video file
mp3_Stream	mp3 stream	Digital audio data stream

3.4.3.4.5. Host Stream Module Processing

3.4.3.5. Microcontroller Communication Manager Description

3.4.3.5.1. Microcontroller Communication Manager Description

The Microcontroller Communication Manager module will be part of our software component running on the server.

3.4.3.5.2. Microcontroller Communication Manager Function

The Microcontroller Communication Manager routes communications to Host Stream Module or Communication/Message Processor module as is appropriate.

3.4.3.5.3. Microcontroller Communication Manager Interfaces

Microcontroller Communication Manager I/O		
Reference #	**IN / OUT**	**Description**
P 3	IN	h.264 Stream, MPG4 Stream, Sensor data and notifications from Microcontroller App Subsystem
P 4	OUT	MP3 Stream and command messages to Microcontroller App Subsystem
VS 3	OUT	h.264 Stream, MPG4 Stream, Sensor data and notifications to Host Stream module
VS 4	IN	MP3 Stream and command messages from Command/Message Processor module
VS 8	OUT	Sensor data and notifications to Command/Message Processor module

3.4.3.5.4. Microcontroller Communication Manager Structure

Microcontroller Communication Manager		
Name	**Type**	**Description**
h624_Stream	h.264 Stream	Digital Video data stream
mpg4_Video	.mpg4 File	Digital video file
door_State	Boolean	Variable to represent the open/close of the Door
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock
lock_Unlock	Integer	Variable to represent the unlock,essage being sent to the Door Lock
lock_Lock	Integer	Variable to represent the lock,essage being sent to the Door Lock
start_Monitor	Integer	Start monitoring message
stop_Monitor	Integer	Stop monitoring message

3.4.3.5.5. Microcontroller Communication Manager Module Processing

```
/*
 * if else statement to route communications
 */
if(communication on P 3 channel){
    if(communication is a stream)
        forward stream to host stream module
    else{
        forward message to
         communications/message processor
       }
}
if(communication is on VS 4 channel)
   forward communication to  Hardware
    Subsystem
```

3.4.3.6. Command/Message Processor Module Description

3.4.3.6.1. Command/Message Processor Module Description

The Command/Message Processor module will be a part of our software component running on the server.

3.4.3.6.2. Command/Message Processor Module Function

The Command/Message Processor module's function is to relay commands between the Android Device and the Smart Door device.

3.4.3.6.3. Command/Message Processor Module Interfaces

Command/Message Processor		
Reference #	**IN / OUT**	**Description**
VS 1	IN	MP3 Stream and command messages from Android Communication Manager module
VS 4	OUT	MP3 Stream and command messages from Command/Message Processor module
VS 7	OUT	Sensor data and notifications to Android Communication Manager module
VS 8	IN	Sensor data and notifications from Microcontroller Communication Manager module

3.4.3.6.4. Command/Message Processor Module Physical Data Structure

Command/Message Processor		
Name	**Type**	**Description**
door_State	Boolean	Variable to represent the open/close of the Door
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock
lock_Unlock	Integer	Variable to represent the unlock,essage being sent to the Door Lock
lock_Lock	Integer	Variable to represent the lock,essage being sent to the Door Lock
start_Monitor	Integer	Start monitoring message
stop_Monitor	Integer	Stop monitoring message

3.4.3.6.5. Command/Message Processor Module Processing

```
/*
 * if else statement to route communications
 */
if(communication on VS 1 channel)
     forward communication to  Microcontroller Communication Manager
if(communication is on VS 8 channel)
   forward communication to  Android Communication Manager Subsystem
```

3.4.3.7. Android Communication Manager Module Description

3.4.3.7.1. Android Communication Manager Module Description

The Android Communication Manager module will be a part of our software component running on the server.

3.4.3.7.2. Android Communication Manager Module Function

The Android Communication Manager module's function is to format messages for communication with the registered Android device.

3.4.3.7.3. Android Communication Manager Module Interfaces

Android Communication Manager		
Reference #	**IN / OUT**	**Description**
P 1	IN	MP3 Stream and command messages from Android App Subsystem
P 2	OUT	h.264 Stream, MPG4 Stream, Sensor data and notifications to Android App Subsystem
VS 1	OUT	MP3 Stream, command messages to the Command Message Processor module
VS 2	IN	h.264 Stream and MPG4 Stream from the Host Stream module
VS 7	IN	Sensor data and notifications from the Command/Message Processor module

3.4.3.7.4. Android Communication Manager Module Physical Data Structure

Android Communication Manager		
Name	**Type**	**Description**
h624_Stream	h.264 Stream	Digital Video data stream
mpg4_Video	.mpg4 File	Digital video file
door_State	Boolean	Variable to represent the open/close of the Door
lock_State	Boolean	Variable to represent the lock/unlock state of the Door Lock
lock_Unlock	Integer	Variable to represent the unlock,essage being sent to the Door Lock
lock_Lock	Integer	Variable to represent the lock,essage being sent to the Door Lock
start_Monitor	Integer	Start monitoring message
stop_Monitor	Integer	Stop monitoring message

3.4.3.7.5. Android Communication Manager Module Processing

Minimal processing,

```
/*
 * if else statement to route communications
 */
if(communication on P 1 channel)
    forward communication to  Command/Message Processor or Host Stream module
if(communication is on VS 2 channel)
   forward communication to  Android GUI Subsystem
```

3.4.3.8. Log Videos/Actions Module Description

3.4.3.8.1. Log Videos/Actions Module Description

The Log Videos/Actions module is a video codec and converter located on the server with the associated logic to execute the codec.

3.4.3.8.2. Log Videos/Actions Module Function

The Log Videos/Actions module will convert the incoming stream to an mp4 file and store it on the server.

3.4.3.8.3. Log Videos/Actions Module Interfaces

Log Videos/Actions		
Reference #	**IN / OUT**	**Description**
V 5	IN	h.264 Stream, MPG4 Stream, Sensor data and notifications from Microcontroller App Subsystem
V 6	IN	MPG4 Stream to Host Stream Subsystem
V 9	OUT	MP3 Stream, Sensor data and notifications from Server HDD Subsystem
N 3	OUT	MPG4 Stream, Sensor data and notifications to Server HDD Subsystem
N 4	IN	MPG4 Stream, Sensor data and notifications from Server HDD Subsystem

3.4.3.8.4. Log Videos/Actions Module Physical Data Structure

Log Videos/Actions		
Name	**Type**	**Description**
mpg4_Video	.mpg4 File	Digital video file
lock_Unlock	Integer	Variable to represent the unlock,essage being sent to the Door Lock
lock_Lock	Integer	Variable to represent the lock,essage being sent to the Door Lock
call_MetaData	Struct	Timestamp, Integer representing alll type, Float for duration

3.4.3.8.5. Log Videos/Actions Module Processing

```
/*
 * Series of if statements to complete log/retrieval of
 * videos and notifications
 */
if(communication is on channel VS 6)
      log the notifications on Server HDD
if(communication  is on channel VS 5)
{
       if(communication is a request for stored video)
             retrieve MPG4 video file from Server HDD
       else if(communication is a video to store on Server HDD)
             store MPG4 video file from Server HDD
}
```

3.4.4. Web Application Subsystem

3.4.4.1. Web Application Subsystem Principles

Usability: The Web Application system must be easy to operate in order for users who may not be technologically sophisticated to be able to use the Smart Door product.

Reliability: The Smart Door device must operate smoothly without fatal crashes.

Security: The Web Application must be hacker resistant.

3.4.4.2. Web Application Subsystem Diagram

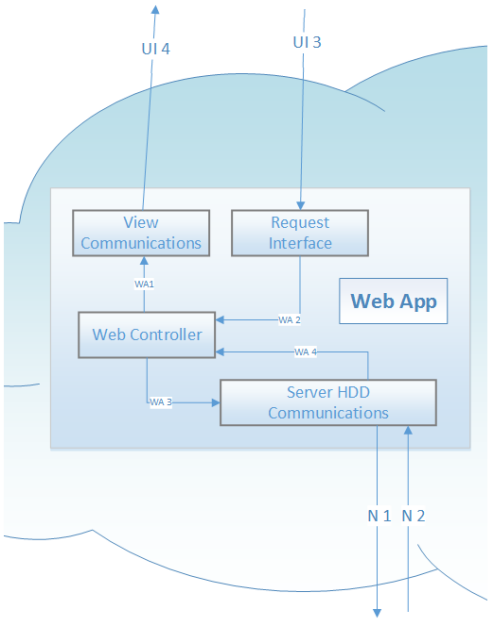


Figure 13: Web Application Subsystem Overview

3.4.4.3. Request Interface Module Description

3.4.4.3.1. Request Interface Description

The Request Interface module is the interface between the Web GUI's incoming request and the Web Application module. The module will parse the request and determine which controller needs to handle the request.

3.4.4.3.2. Request Interface Function

The main function of the Request Interface is to parse the request and the data in it and pass it to the controller in an order that the controller knows how to use.

3.4.4.3.3. Request Interface Module Interfaces

Request Interface Interfaces		
Reference #	**IN / OUT**	**Description**
UI3	IN	HTTP request from the Web GUI
WA2	OUT	Data object containing a list of variables.

3.4.4.3.4. Request Interface Module Physical Data Structure

Request Interface Data		
Name	**Type**	**Description**
HTTP String	HTTP	A string with the controller, method, and variable that need to be executed
Request data	RequestData	An object that holds the request variables

3.4.4.3.5. Request Interface Module Processing

```
/*
 * parse request variables
 * get controller
 * get method
 * call controller method with variables
 */
RequestInterface( http ) {
        variablesList = pareseRequest(http);
        controller = getController(http);
        method = getMethod(http);
        WebController(controller, method, variablesList);
       }
```

3.4.4.4. Web Controller Module Description

3.4.4.4.1. Web Controller Description

The Web Controller module runs C-Sharp code to accomplish predefined task on the server. The web controller will receive a controller name, method name, and variables. The controller will execute a predefine C Sharp function which will use the variables passed by the View Communications module.

3.4.4.4.2. Web Controller Function

The main function of the Web Controller is to retrieve and store data in the server. The most important part of this module is the predefine controller files it executes. The file it executes enables the user to register devices to account, monitor, download video, delete video, login/logout, create web account, and generate a mobile key.

3.4.4.4.3. Web Display Module Interfaces

Web Controller Interfaces		
Reference #	**IN / OUT**	**Description**
WA1	OUT	Object with the data need to construct the view
WA2	IN	Variables need to run the controller method
WA3	OUT	SQL statement that needs to be validated
WA4	IN	Response of a SQL statement

3.4.4.4.4. Web Controller Module Physical Data Structure

Web Controller Data		
Name	**Type**	**Description**
Variables	ALL Data Types	Function parameters converted from strings to an arbitrary data type
SQLStatement	String	A string which will be sent to the Server HHD Communications
ListOfData	List	A list of instructions that will be sent to the Web GUI
Register Device to Account	CS File	The register device to account file is triggered by a request from the Web GUI to store an association in the SQL database. The file will validate that the model for the data structure is correct and that it is a safe SQL variable.
Monitor	CS File	The monitor file takes the request for a stream of a video file stored in the Server HHD to the Web GUI.
Download Video	CS File	The file will send the Web GUI the video file it has requested from the Server HHD.
Delete Video	CS File	The file will remove a video file from the Server HHD
Login/Logout	CS File	The file authorizes and de authorizes users of the system. The file will hash the password inputted and apply the salt of the username inputted and compare the hashed and salted passwords together. If the password combination works then it creates a session for the user and grants the user access to the system and his information.
Create Web Account	CS File	The file allows new user to create accounts with the system. The file will need to make sure that the password is of sufficient length and format. The file will need to make sure that the username enters is unique to the system.
Generate Mobile Key	CS File	The file will generate and map a GUID to identify the mobile devices paired with this account, and store the association in the Server HHD.

3.4.4.4.5. Web Controller Module Processing

```
/*
 * read C-sharp file
 * run file
 * if successful pass instructions to view communications
 */
WebController( controller, method, variables ) {
        ControllerObject = Open(controller);
        Instructions = ControllerObject.run(method, variables);
        If(Instructions.success){
          ViewCommunications(Instructions);
        }

       }
```

3.4.4.5. Server HHD Communications Module Description

3.4.4.5.1. Server HHD Communications Description

The Server HHD Communications module is the interface between the Web Application and the Server HHD subsystem. The module will parse the SQL Statement and determine if it follows the correct syntax. The module will also make sure that the variables are not harmful and send the command to be executed.

3.4.4.5.2. Server HHD Communications Function

The main function of the Server HHD Communications is to parse the SQL and the data in it, validate it, and hand it to the Server HHD in an order to execute a SQL command.

3.4.4.5.3. Server HHD Communications Module Interfaces

Server HHD Communications Interfaces		
Reference #	**IN / OUT**	**Description**
WA3	IN	SQL Command for the controller
WA4	OUT	Execution response
N1	OUT	Valid SQL Statement
N2	IN	Execution response

3.4.4.5.4. Server HHD Communications Module Physical Data Structure

Server HHD Communications Data		
Name	**Type**	**Description**
SQL String	String	A string with the SQL Statement
SQL Variables	All Data Types	A list of variables

3.4.4.5.5. Server HHD Communications Module Processing

```
/*
  * validate statement and variables
  * if valid execute command
  */
ServerHHDCommunications( SQL, Variables ) {
            Command = sqlValid(SQL,Variables)
            If(Command.valid){
              ServerHHD(Command)
            }
          }
```

3.4.4.6. View Communications Module Description

3.4.4.6.1. View Communications Description

The View Communications module is the interface between the Web Application outgoing request and the Web GUI subsystem. The module will transfer the instruction the controller created to the Web GUI.

3.4.4.6.2. View Communications Function

The main function of the View Communications is to pass the data received from the Web controller to the Web GUI, and to validate that the data has the correct format and syntax.

3.4.4.6.3. View Communications Module Interfaces

View Communications Interfaces		
Reference #	**IN / OUT**	**Description**
UI4	OUT	Valid Instructions object
WA1	IN	List of instructions

3.4.4.6.4. View Communications Module Physical Data Structure

View Communications Data		
Name	**Type**	**Description**
Instructions	Instructions	An object containing the view, and partial layouts that will create the HTML file, and the data need to execute the views

3.4.4.6.5. View Communications Module Processing

```
/*
 * validate data loop
 * send data
 */
ViewCommunications(Instructions) {
        For each object in Instructions
            If valid instruction
              Continue
            Else
               Return 0
          sendInstructions(Instructions);
        }
```

3.5.Data Storage Layer

3.5.1. Data Storage Layer Overview

Figure 14: Data Storage Layer Overview

3.5.2. Data Storage Layer Data Element Description

No new inter-layer data communications to describe.

3.5.3. Server HDD Subsystem

3.5.3.1. Server HDD Subsystem Principles

Security: Digital and physical security measures on the Microcontroller HDD will help to keep the users data safe.
Interactive: No interactive functionality of the Smart Door system is possible without somewhere to store the data supporting that functionality.
Reliability: The overall Smart Door performance relies strongly on the reliability of the Microcontroller HDD. Any delay or failure of this system will completely disable all functionality of the Smart Door system.
Usability: The Smart Door system will not be usable as anything other than a doorstop without somewhere to store the system data.

3.5.3.2. Server HDD Subsystem Diagram

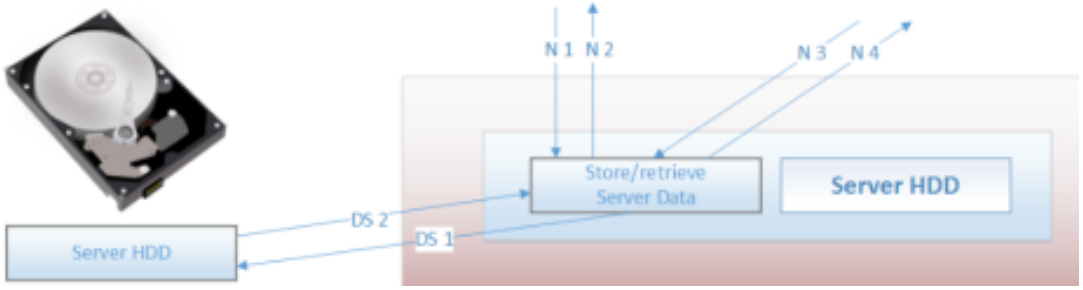


Figure 15: Server HDD Subsystem Overview

3.5.3.3. Store/Retrieve Server Data Module Description

The Server HHD stores the Server OS, configuration data, and online data. The Server HHD Stores all data backups when they are uploaded by the system.

3.5.3.4. Retrieve Server Data Module Function

Retrieve HDD data module's functionality is handled natively by the Linux OS.

3.5.3.5. Retrieve Server Data Module Interfaces

Store/Retrieve Server Data I/O		
Reference #	**IN / OUT**	**Description**
N 1	IN	MPG4 Stream, Sensor data and notifications from the Web App Subsystem
N 2	OUT	MPG4 Stream, Sensor data and notifications to the Web App Subsystem
N 3	IN	MPG4 Stream, Sensor data and notifications from the Video Server Subsystem
N 4	OUT	MPG4 Stream, Sensor data and notifications to the Video Server Subsystem
DS 1	OUT	MPG4 Stream, Sensor data and notifications to the Server HDD
DS 2	IN	MPG4 Stream, Sensor data and notifications from the Server HDD

3.5.3.6. Retrieve Server Data Module Physical Data Structure

Store/Retrieve Server Data Physical Data Structure		
Name	**Type**	**Description**
mpg4_Video	.mpg4 File	Digital video file
satus_Notify	Struct	Timestamp, Integer representing a particular status
call_MetaData	Struct	Timestamp, Integer representing call type, Float for duration of call

3.5.3.7. Retrieve Server Data Module Processing

3.5.4. Micro Controller HDD Subsystem

3.5.4.1. Micro Controller HDD Subsystem Principles

Security: Digital and physical security measures on the Server HDD will help to keep the users data safe.
Interactive: No interactive functionality of the Smart Door system is possible without somewhere to store the data supporting that functionality.
Reliability: The overall Smart Door performance relies strongly on the reliability of the Server HDD. Any delay or failure of this system will completely disable all functionality of the Smart Door system.
Usability: The Smart Door system will not be usable as anything other than a doorstop without somewhere to store the system data.

3.5.4.2. Micro Controller HDD Subsystem Diagram

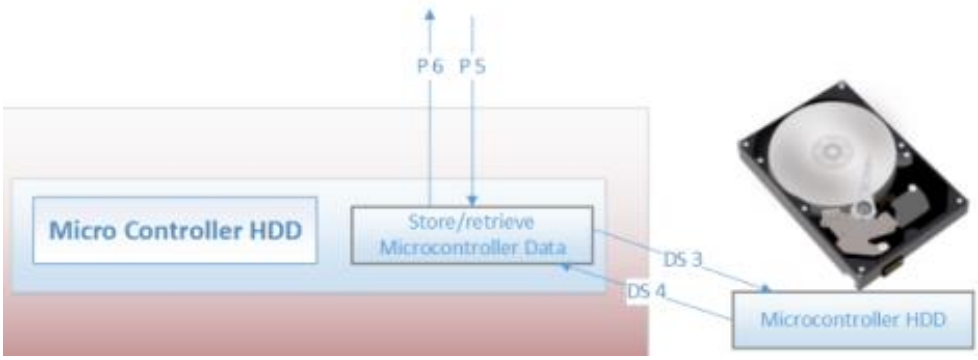


Figure 16: Microcontroller HDD Subsystem Overview

3.5.4.3. Store/Retrieve HDD Data Module Description

The Microcontroller HHD stores the microcontroller OS, configuration data, and offline data. The Microcontroller HHD Stores all data backups when there is a lack of WIFI signal.

3.5.4.4. Retrieve HDD Data Module Function

Retrieve HDD data module's functionality is handled natively by the Linux OS.

3.5.4.5. Retrieve HDD Data Module Interfaces

Microcontroller HDD		
Reference #	**IN / OUT**	**Description**
P 5	IN	MPG4 Stream, Sensor data and notifications from the Microcontroller Application Subsystem
P 6	OUT	MPG4 Stream, Sensor data and notifications to the Microcontroller Application Subsystem
DS 3	OUT	MPG4 Stream, Sensor data and notifications to the Microcontroller HDD
DS 4	IN	MPG4 Stream, Sensor data and notifications from the Microcontroller HDD

3.5.4.6. Retrieve HDD Data Module Physical Data Structure

Microcontroller HDD		
Name	**Type**	**Description**
mpg4_Video	.mpg4 File	Digital video file
satus_Notify	Struct	Timestamp, Integer representing a particlar status
call_MetaData	Struct	Timestamp, Integer representing call type, Float for duration of call

3.5.4.7. Retrieve HDD Data Module Processing

4. Quality Assurance

4.1. Overview

4.1.1. Purpose

The Quality Management Plan (QMP) is necessary to ensure product design and implementation meets specifications. It will be used as a way to validate and verify our product solves the original problem given. This includes our requirements and those of our stakeholders.

4.1.2. Requirements Analysis and Review

This is a continual review process that ensures that specified requirements are still feasible and within the scope of our design. The SRS requirements are not engraved in stone. The team will constantly monitor our requirements and receive stakeholder feedback to ensure product viability.

4.1.3. Feasibility

This is similar to the requirements analysis and review, except this focuses on our team's ability to meet project requirements. If the team is unable to deliver a product that meets its specifications then we will need to determine how feasible the feature is that is not meeting requirements. This is a combination of Risk Management, but requires a QMP for qualitative and quantitative analysis.

4.1.4. Documentation

Documentation is critical to the success of our product. It not only increases the longevity of the product by allowing other developers to extend its capabilities, but it also acts as a risk mitigation tool for identifying design and implementation mistakes. Our documentation should be of such quality that all members on the team are able to discern each other's design from their description.

4.1.5. Coding Review and Analysis

The coding review will help the team in two areas. First, it will allow additional team members to verify and validate the implementation to reduce errors and confusion. Second, we will all have knowledge of the system, which will make our design process more cohesive. It essentially serves to reinforce our product design.

4.2.Key Test Assumptions/Requirements/dependencies

4.2.1. Module/Unit Level Testing

Module and Unit level testing are iterative processes. We will conduct module level testing upon the completion of each module and we will conduct unit level testing upon the completion of related modules. Module level testing will be defined as validating and verifying the internal functions within each module upon the completion of implementing the module. Unit testing will be defined as testing the interfaces and data transfer between two related modules. Modules are defined as related if they exchange any messages or information between each other. Module and Unit level testing will not include interfaces across subsystem boundaries. These interfaces will be tested during Integration testing.

4.2.1.1. Test Assumptions

4.2.1.1.1. Module level testing assumes that the module has been completely implemented and debugged.

4.2.1.1.2. Unit level testing assumes that each participating module has been implemented and debugged.

4.2.1.2. Test Requirements

Requirements will be generated for each module based on the system level requirements they are intended to fulfill according to the requirements traceability matrix. Additional functional requirements will assure that the module fulfills its function within the subsystem.

4.2.1.3. Test Dependencies

4.2.1.3.1. The Module level tests are intended to have no external dependencies since they are only testing internal functionality of a module.

4.2.1.3.2. Unit level testing will depend on the proper implementation of each participating module. Additionally, unit level testing will not complete successfully unless each participating module adheres to its interface requirements.

4.2.2. Component Level Testing

Component level testing will test the functionality and internal data communication of each individual subsystem. Component level testing will not include interfaces across subsystem or architectural layer boundaries. These interfaces will be tested during Integration testing.

4.2.2.1. Test Assumptions

4.2.2.1.1. Component level testing assumes that each module within the subsystem has passed module level testing.

4.2.2.1.2. Component level testing assumes that each subset of related modules within a subsystem has passed unit level testing.

4.2.2.2. Test Requirements

Requirements will be generated for each subsystem based on the system level requirements they are intended to fulfill according to the requirements traceability matrix. Additional functional requirements will assure that the subsystem fulfills its function within the overall system.

4.2.2.3. Test Dependencies

4.2.2.3.1. Component level tests are intended to test the subsystem's internal functionality. This means that the component level tests are dependent on the module and unit level tests being passed successfully.

4.2.2.3.2. Component level tests depend on each module adhering to its defined interface requirements in order for communication testing to complete successfully.

4.2.3. Integration Testing

4.2.3.1. Test Assumptions

4.2.3.2. Test Requirements

4.2.3.3. Test Dependencies

4.2.4. System Verification Testing

Verification testing will confirm that Always Home designed and implemented the Smart Door system in such a way that the system as a whole meets all of the system level requirements. Validation confirms that the Smart Door meets the needs of the stakeholders and will occur upon completion of the Acceptance criteria outlined in section six of this document.

4.2.4.1. Test Assumptions

4.2.4.2. Test Requirements

4.2.4.3. Test Dependencies

4.3.Brief Test Case Description for Function Testing

4.3.1. Expanded in Test Plan

The functional testing plan will be defined in the Test Plan document to be delivered with the prototype.

5. Requirements Traceability Matrix

5.1. Requirements Traceability Matrix Purpose

Team Always Home will use the requirements traceability matrix for two purposes, requirements verification and system analysis. For requirements verification, the matrix verifies that all requirements are being fulfilled and that there are no unused modules. For system analysis, the matrix helps to identify modules which are overly complex, have high coupling, or other issues. The requirements verification task identifies particular requirements which are not being met by the currently defined system architecture and it also identifies components that are not being used by the currently defined system architecture to fulfill any requirements. This is very valuable for verifying and simplifying the system.

5.2. System Level Requirements Traceability Matrix

Requirements Traceability Matrix									
	Ui Layer			Processing Layer		Network Layer		Data Storage Layer	
	Hardware	Web Gui	Android Gui	Microcontroller App	Android App	Web App	Video Server	Server HDD	Microcontroller HDD
Customer Requirements									
Smartphone Application Control			X		X				
Snapshots Taken	X								
Keep Activities Log	X						X	X	X
Emergency Power Supply									
Video Monitoring	X		X		X		X		
System Portability	X								
Smart Phone Paring									
Motion Sensor									
Local Storage	X								X
Hardware Security	X								
Z-Wave									
Lock/Unlock	X		X		X		X		
Open Source	X			X	X	X	X		
Nonintrusive	X								
Packaging Requirements	Microcontroller	Web Gui	Android Gui	Microcontroller App	Android App	Web App	Server OS	Server HDD	MC HDD
Size	X								
Temperature Control	X								
Mounting	X								
Casing	X								
Software Acquisition			X		X				
Included Components	X								
Team Logo	X								
System Configuration	X	X	X				X		
Performance Requirements	Microcontroller	Web Gui	Android Gui	Microcontroller App	Android App	Web App	Server OS	Server HDD	MC HDD
System Setup	X			X	X	X	X		
Notification Time	X			X	X	X	X		
Latency	X			X	X	X	X		
Response Time	X	X	X	X	X	X	X	X	X
Data Transmission - Live Video Feed	X			X	X		X		
Data Transmission - Audio Transmission	X			X	X		X		
Recording Log - Log Availability		X				X	X	X	X
Recording Log - Storage Capacity	X							X	X
Power - Power Supply	X								
Power - Power Source	X								
Emergency Backup Battery									
I/O Ports	X								
Operating Environment									
Account Setup		X	X	X	X	X	X		
Video Log Downloads				X	X	X	X	X	
Video/Photo Resolution	X								
System Availability	X			X	X	X	X		
Notification Limit	X			X	X		X		
Initialization Time	X			X	X	X	X		
Mounting Height	X								
Microphone Sensitivity	X				X				
Safety Requirements	Microcontroller	Web Gui	Android Gui	Microcontroller App	Android App	Web App	Server OS	Server HDD	MC HDD
Camera Mounting	X								
Microphone Mounting	X								
Camera and Microphone Weather Safety	X								
Receptacles	X								
Maintenance and Support Requirements	Microcontroller	Web Gui	Android Gui	Microcontroller App	Android App	Web App	Server OS	Server HDD	MC HDD
Software Updates				X	X	X			
User Manual				X	X	X			
Open Source				X	X	X			
Other Requirements	Microcontroller	Web Gui	Android Gui	Microcontroller App	Android App	Web App	Server OS	Server HDD	MC HDD
Web Interface Support		X							
Portable Source Code				X	X	X			
Notification Control					X				

Table 9: Requirements Traceability Matrix

5.2.1. System Level Requirements Traceability Matrix Analysis

Always Home used the Requirements Traceability matrix in order to verify that all requirements had been fulfilled and prioritize development order of subsystems and modules. As shown in the Requirements Traceability Matrix above, the UI layer contributes to the fulfilment of forty-one requirements. As the entire project is only subject to fifty requirements this means that the UI layer must support eighty two percent of the projects requirements. This would indicate that

even though UI may not be critical for the projects functionality it is quite important for the customers' acceptance of the Smart Door.

5.2.2. System Level Requirements Traceability Matrix Findings

Based on our analysis we will prioritize our development as described below. First we will develop initial functionality of the Microcontroller Application, Android Application and Video Server subsystems. This will allow us to verify video and audio streaming capabilities early in our implementation phase of development. Following the verification of a working data path and video streaming capabilities we will continue to refine these modules and develop the remaining modules.

5.3. UI Layer Requirements Traceability matrix

UI Layer Requirements Traceability Matrix

	Web GUI			Android Gui			Hardware			
	Web Display	View Constructor	Request Handler	Display/Layout Manager	Android Event Handler	Android Function Handler	USB Interface	Microcontroller BUS	GPIO Interace	Speaker Interface
Customer Requirements										
Smartphone Application Control					X	X				
Snapshots Taken							X	X		
Keep Activities Log								X		
Emergency Power Supply										
Video Monitoring				X	X	X	X	X		
System Portability								Requirement Satisfied by Smart Door Device		
Smart Phone Paring										
Motion Sensor										
Local Storage							X	X		
Hardware Security								Requirement Satisfied by Smart Door Device		
Z-Wave										
Lock/Unlock					X	X		X	X	
Open Source								X		
Nonintrusive								Requirement Satisfied by Smart Door Device		
Packaging Requirements										
Size								Requirement Satisfied by Smart Door Device		
Temperature Control								Requirement Satisfied by Smart Door Device		
Mounting								Requirement Satisfied by Smart Door Device		
Casing								Requirement Satisfied by Smart Door Device		
Software Acquisition					X	X				
Included Components							X	X	X	X
Team Logo								Requirement Satisfied by Smart Door Device		
System Configuration			X		X	X	X	X	X	X
Performance Requirements										
System Setup								X		
Notification Time								X		
Latency							X	X	X	X
Response Time	X	X	X	X	X	X	X	X	X	X
Data Transmission - Live Video Feed							X	X		
Data Transmission - Audio Transmission							X	X		X
Recording Log - Log Availability			X							
Recording Log - Storage Capacity								Requirement Satisfied by Smart Door Device		
Power - Power Supply								Requirement Satisfied by Smart Door Device		
Power - Power Source								Requirement Satisfied by Smart Door Device		
Emergency Backup Battery										
I/O Ports							X	X	X	X
Operating Environment										
Account Setup			X		X	X				
Video Log Downloads										
Video/Photo Resolution							X	X		
System Availability							X	X	X	X
Notification Limit								X		
Initialization Time							X	X	X	X
Mounting Height								Requirement Satisfied by Smart Door Device		
Microphone Sensitivity								X		X
Safety Requirements										
Camera Mounting								Requirement Satisfied by Smart Door Device		
Microphone Mounting								Requirement Satisfied by Smart Door Device		
Camera and Microphone Weather Safety								Requirement Satisfied by Smart Door Device		
Receptacles								Requirement Satisfied by Smart Door Device		
Maintenance and Support Requirements										
Software Updates										
User Manual										
Open Source										
Other Requirements										
Web Interface Support	X	X	X							
Portable Source Code										
Notification Control										

Table 10: UI Layer Requirements Traceability Matrix

5.3.1. UI Layer Requirements Traceability Matrix Analysis

The UI Layer is critical for the Customer and Performance categories of requirements. The modules in this layer support the fulfillment of forty-three requirements. As the entire system has fifty three requirements, the UI layer supports approximately eighty percent of the requirements in the system..

5.3.2. UI Layer Requirements Traceability Matrix Findings

Within this layer a Three issues that stand out as areas of concern. Firstly, every module contributes to Response time which implies that it will be very hard to succeed in meeting our goals. Since the response time will be shared amongst many modules the variance per module must be extremely low in order to keep the overall variance within our goal. The second concern is that the microcontroller bus has too many responsibilities. Since all the communications must flow though this module we must take care during development to assure that the timing and structure of the communications does not exceed the bandwidth of the BUS. Finally many of the requirements are marked as satisfied by the Smart Door device, this implies that the requirements will only be satisfied upon installation of the device. Unfortunately, this means that Always Home will be unable to verify these requirements in any meaningful way. We can mount the device to show that is possible to fulfill the requirement with the prototype but for the requirement to be satisfied for any given end user they must be able to follow directions and install the device correctly.

5.4. Processing Layer Requirements Traceability matrix

Processing Layer Requirements Traceability Matrix

	Android App					Microcontroller App				
	Android Activity	Push Notifications	Android Command Interpreter	Video Stream	Server Communication Manager	Data Interpreter	Data Processor	Startup Manager	Server Communication Manager	Local Data Access
Customer Requirements										
Smartphone Application Control	X		X		X					
Snapshots Taken										
Keep Activities Log										
Emergency Power Supply										
Video Monitoring	X		X	X	X					
System Portability										
Smart Phone Paring										
Motion Sensor										
Local Storage										
Hardware Security										
Z-Wave										
Lock/Unlock	X		X		X					
Open Source	X	X	X	X	X	X	X	X	X	X
Nonintrusive										
Packaging Requirements										
Size										
Temperature Control										
Mounting										
Casing										
Software Acquisition					X					
Included Components										
Team Logo										
System Configuration										
Performance Requirements										
System Setup	X		X		X	X	X	X	X	X
Notification Time	X	X	X		X	X	X		X	X
Latency	X		X		X	X	X		X	X
Response Time	X		X		X	X	X		X	X
Data Transmission - Live Video Feed	X		X	X	X	X	X		X	
Data Transmission - Audio Transmission	X		X		X	X	X		X	
Recording Log - Log Availability										
Recording Log - Storage Capacity										
Power - Power Supply										
Power - Power Source										
Emergency Backup Battery										
I/O Ports										
Operating Environment										
Account Setup	X		X		X		X		X	X
Video Log Downloads	X		X		X		X		X	X
Video/Photo Resolution										
System Availability	X		X		X	X	X	X	X	X
Notification Limit	X	X				X	X		X	X
Initialization Time	X		X		X	X		X	X	X
Mounting Height										
Microphone Sensitivity	Requirement Satisfied by Android Device									
Safety Requirements										
Camera Mounting										
Microphone Mounting										
Camera and Microphone Weather Safety										
Receptacles										
Maintenance and Support Requirements										
Software Updates	X	X	X	X	X	X	X	X	X	X
User Manual						X				
Open Source	X	X	X	X	X	X	X	X	X	X
Other Requirements										
Web Interface Support										
Portable Source Code	X	X	X	X	X	X	X	X	X	X
Notification Control	X	X								

Table 11: Processing Layer Requirements Traceability Matrix

5.4.1. Processing Layer Requirements Traceability Matrix Analysis

The processing layer requirements traceability matrix is the densest matrix created during the analysis of requirements. This implies that much of the development effort will be focused in this area. Of the

requirements satisfied within this layer only the Lock/Unlock requirement is satisfied by a single module. The rest of the requirements are satisfied jointly by three to ten modules.

5.4.2. Processing Layer Requirements Traceability Matrix Findings

The dense nature of this table indicates how important the proper development of the Processing Layer is to the success of the Smart Door. The main issue here is the high level of interrelatedness between the modules of the Processing layer. During the module and subsystem design Always Home strove to imbed simple and logical data flows in order to reduce the complexity of the system. Unfortunately the distributed nature of the system was working against us achieving our goal. This matrix illustrates how the problem of high coupling manifested itself in the Processing Layer. Although the modules process and manipulate data in isolation from each other, their extensive communication structures re-introduce the problems we sought to avoid by dividing the system the way we did.

5.5. Network Layer Requirements Traceability matrix

Network Layer Requirements Traceability Matrix

	Web App				Video Server				
	View Communications	Request Interface	Web Controller	Server HDD Communications	Android Communication Manager	Microcontroller Communication Manager	Host Stream	Command/Message Processor	Log Videos/Actions
Customer Requirements									
Smartphone Application Control									
Snapshots Taken									
Keep Activities Log					X	X		X	X
Emergency Power Supply									
Video Monitoring					X	X	X	X	
System Portability									
Smart Phone Paring									
Motion Sensor									
Local Storage									
Hardware Security									
Z-Wave									
Lock/Unlock					X	X		X	X
Open Source	X	X	X	X	X	X	X	X	X
Nonintrusive									
Packaging Requirements									
Size									
Temperature Control									
Mounting									
Casing									
Software Acquisition									
Included Components									
Team Logo									
System Configuration					X	X		X	
Performance Requirements									
System Setup	X	X	X	X	X	X		X	
Notification Time					X	X		X	X
Latency					X	X	X	X	
Response Time					X	X	X	X	
Data Transmission - Live Video Feed					X	X	X		
Data Transmission - Audio Transmission					X	X	X		
Recording Log - Log Availability	X	X	X	X	X				X
Recording Log - Storage Capacity									
Power - Power Supply									
Power - Power Source									
Emergency Backup Battery									
I/O Ports									
Operating Environment									
Account Setup	X	X	X	X	X	X		X	
Video Log Downloads	X	X	X	X	X	X		X	X
Video/Photo Resolution									
System Availability	X	X	X	X	X	X	X	X	X
Notification Limit					X	X		X	
Initialization Time					X	X		X	
Mounting Height									
Microphone Sensitivity									
Safety Requirements									
Camera Mounting									
Microphone Mounting									
Camera and Microphone Weather Safety									
Receptacles									
Maintenance and Support Requirements									
Software Updates	X	X	X	X					
User Manual									
Open Source	X	X	X	X	X	X	X	X	X
Other Requirements									
Web Interface Support									
Portable Source Code	X	X	X	X					
Notification Control									

Table 12: Network Layer Requirements Traceability Matrix

5.5.1. Network Layer Requirements Traceability Matrix Analysis

Although the Network Layer satisfies fewer requirements than the UI Layer, it still contributes to the satisfaction of over forty percent of the requirements levied on the smart door. This layer works closely with the Processing Layer and has a similar issue with each requirements being satisfied by four to nine modules. This is an extremely high level of cohesion that will have to be closely watched during implementation of the project.

5.5.2. Network Layer Requirements Traceability Matrix Findings

The nature of the network layer is that it facilitates the communication and data flows between the disparate parts of the Smart Door system. It is this aspect of the system design that causes this layer to be so critical to the success of the project. Once again the modules in the Network Layer are tightly coupled by their data flows and great care must be taken during implementation phase to keep the various modules independent.

5.6. Data Storage Layer Requirements Traceability matrix

Data Storage LayerRequirements Traceability Matrix				
	Server HDD		Microcontroller HDD	
	Store/Retrieve Server Data	Server HDD	Store/Retrieve Microcontorller Data	Microcontroller HDD
Customer Requirements				
Smartphone Application Control				
Snapshots Taken				
Keep Activities Log	X	X	X	X
Emergency Power Supply				
Video Monitoring				
System Portability				
Smart Phone Paring				
Motion Sensor				
Local Storage			X	X
Hardware Security				
Z-Wave				
Lock/Unlock				
Open Source				
Nonintrusive				
Packaging Requirements				
Size				
Temperature Control				
Mounting				
Casing				
Software Acquisition				
Included Components				
Team Logo				
System Configuration				
Performance Requirements				
System Setup				
Notification Time				
Latency				
Response Time	X	X	X	X
Data Transmission - Live Video Feed				
Data Transmission - Audio Transmission				
Recording Log - Log Availability	X	X	X	X
Recording Log - Storage Capacity	X	X	X	X
Power - Power Supply				
Power - Power Source				
Emergency Backup Battery				
I/O Ports				
Operating Environment				
Account Setup				
Video Log Downloads	X	X		
Video/Photo Resolution				
System Availability				
Notification Limit				
Initialization Time				
Mounting Height				
Microphone Sensitivity				
Safety Requirements				
Camera Mounting				
Microphone Mounting				
Camera and Microphone Weather Safety				
Receptacles				
Maintenance and Support Requirements				
Software Updates				
User Manual				
Open Source				
Other Requirements				
Web Interface Support				
Portable Source Code				
Notification Control				

Table 13: Data Storage Layer Requirements Traceability Matrix

5.6.1. Data Storage Layer Requirements Traceability Matrix Analysis

As is evident by the sparseness of this table the storage layer is both simple and contributes to the satisfaction of very few requirements. The simplicity of these modules does not mean that they are not important. The requirements satisfied by the HDDs and the functionality they provide are both crucial to the overall systems performance and usability.

5.6.2. Data Storage Layer Requirements Traceability Matrix Findings

The HDDs are simply connected hardware and all specified functionality is supported by drivers installed with the operating system.

6. Acceptance Plan Assumptions Relative to Detailed Design

6.1.Packaging and Installation

6.1.1. RTMP Server installation

The RTMP server will be automatically installed as part of the overall software install and the installer will prompt the user for any information that cannot be obtained through Windows APIs.

6.2.Acceptance Testing

Each Acceptance Criteria will have a unique functional test to be carried out by a sponsor representative. Successful completion of all ten acceptance tests will constitute acceptance of the product. These tests will be specified in the Smart Door test plan.

6.3.Acceptance Criteria

Criteria Number	Criterion Description
9.01	Wakes and notifies upon connected doorbell being pressed
9.02	Wakes and logs the status of the connected door upon connected door opening or closing
9.03	Live video monitoring capability within the Smart Door android app
9.04	One way video communication between door and the Smart Door android app
9.05	Two way audio communication between door and Smart Door android app
9.06	Smart Door system returns to sleep mode upon completion of interaction
9.07	Video logging and ability to retrieve videos using the Smart Door web app
9.08	Smart Door system portability and ease of installation
9.09	Smart Door system does not interfere with normal operation of door
9.10	Smart Door system supports multiple Smartphones

Table 14: Acceptance Criteria

7. Appendices

7.1. N/A

No Appendices exist for this document